SPECIAL GRAPHICS ISSUE

FDC 50078

ST LOG

THE ATARI ST MONTHLY MAGAZINE

JULY 1989 ISSUE 34

DISK VERSION $12.95

MICROCHECK ST

DRAW IT!

ANIMATED INPUT

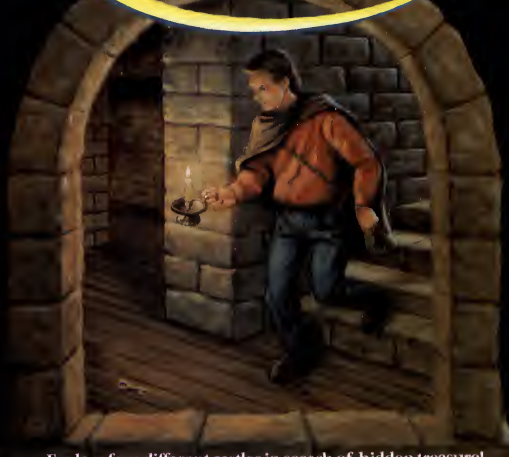
REVIEWS:

Touch-Up
LDW Power
Prospero Pascal
Tower Toppler
And More!

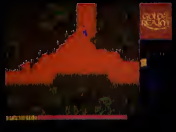
07
0 74369 50078 5

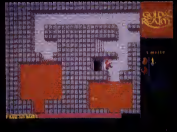
For Atari ST Series Computers

GOLD of the REALM



• Explore four different castles in search of hidden treasure!
• Over 300 different full color screens. • Three degrees of difficulty.





P.O. Box 17422
Phoenix, AZ 85011
(602) 265-7849

Magnetic Images™

CIRCLE #101 ON READER SERVICE CARD.

EDITORIAL

BY ANDY EDDY
ASSOCIATE · EDITOR

As I write this, COMDEX (Computer Dealers' Exposition) is taking place in Chicago. This is where all computer-related manufacturers display their wares for buyers and media people, and the news is that Atari is there in force to promote their products.

Atari has had its ups and downs at these computer shows: In better times they would find their way into the "highlight films" of journalists' coverage, while at other times they wouldn't even attend the show because they had nothing new to announce and felt their money was better spent elsewhere.

This COMDEX brings word of *new* Atari products, a welcome change of the past few years. One of the products, Folio, is an attempt to add some spice to the MS-DOS world by creating an inexpensive, hand-held IBM-compatible computer—"about the size of a Walkman," as the press release reads. This refines the concept of a laptop, allowing salesmen and other on-the-road executives to easily bring a computer with them on their travels.

The more exciting COMDEX news for long-time Atari fans, though, is the showing of Stacy, the ST laptop. Although isolated within a glass case to protect it from the hands of passers-by, it is actually there for *all* to see—unlike their invitation-only showings of the past. Atari is also pitching a DTP (desktop publishing) system, which includes an ST, a Postscript-compatible laser printer (a plus for compatibility with other brands of computers and software), a hard drive and a cache of 35 fonts. Certainly this is good news for all Atari owners, as it appears to be a sign of Atari's return to serious U.S. marketing.

Along those lines, the word from the rumor mill—which has been unnervingly silent since Atari altered their marketing stance by maintaining silence about products until they are ready to ship—tells of the TT, the enhanced ST computer. Though it's too early to venture exactly what will be under the hood, Atari has proven their knack for innovation, even if they seem to lack crucial insight on marketing their products after they hit the shelves.

Another exciting development is a revolutionary MIDI keyboard being developed by Atari with the help of Mick Fleetwood, one of the leaders behind the rock group Fleetwood Mac. It's rumored to be so intuitive that it may change the method and speed with which people learn music. As the ST-LOG staff consists of many musicians, both professional and amateur, we're pretty excited about the possibilities.

Of course, as these products are released and hit store shelves, ST-LOG will be here to present evaluations for you, so you can best decide if they are something you'd like to add to your computer den. In the meantime, we sit on our hands in anxious anticipation of the "next generation" machine's appearance on our doorsteps.■

IN THIS ISSUE

FEATURES

REVIEWS

COLUMNS

DEPARTMENTS

PROGRAM · LISTING · GUIDE

ST-LOG (ISSN 0890-9601) is published monthly by L.F.P., inc., 9171 Wilshire Blvd., Suite 300, Beverly Hills, CA 90210.

U.S. subscription $28 for one year (12 issues), $52 for two years (24 issues), $76 for three years (36 issues). Foreign subscriptions: add $10 per subscription per year. Single copy $3.50 (add $1 for postage). Change of address: six weeks advance notice, and both old and new addresses are needed. POSTMASTER, send change of address to: **ST-LOG** Magazine, P.O. Box 16928, North Hollywood, CA 91615. Second-class postage paid at Beverly Hills, CA, and additional mailing offices.

JULY 1989
ISSUE 33

STAFF

Publieher: Lee H. Pappas. **Executive Editor:** Clayton Walnum. **Aeeociate Editor:** Andy Eddy. **Art Director:** Andy Dean. **Meneging Editor:** Dean Brierly. **Eeet Coeet Editor:** Arthur Leyenberger. **Weet Coaet Editor:** Charles F. Johnson. **Contributing Editors:** Ian Chadwick, Frank Cohen, Maurice Molyneaux, Steve Panak, Marshal M. Rosenthal, George L. Smyth. **Copy Chief:** Katrina Veit. **Copy Editors:** Sarah Bellum, Norma Edwards, Randolph Heard, Pat Romero, Kim Turner. **Chief Typographer:** Alice Nichols. **Typographers:** David Buchanan, B. Miro Jr., Quita Saxon. **Editoriel Aseietant:** Patricia Koury. **Contributore:** Albert Baggetta, D.A. Brumleve, Frank Eva, Robert Golf, Mark E. Nelson, Robert Osness, David Plotkin, Karl E.Wiegers. **Vice President, Production:** Donna Hahner. **Production Aeeieteni:** Steve Hopkins. **National Advertising Director:** Jay Eisenberg. **Corporate Director of Advertising:** Paula S. Thornton. **Advertieing Production Director:** Janice Rosenblum. **Advertieing Production Coordinator:** Maggie Chun. **Subecriptions Director:** Irene Gradsten. **Vice President, Selee:** James Gustafson.

U.S. newsstand distribution by Eastern News Distributors, Inc., 1130 Cleveland Rd., Sandusky, OH 44870.

ADVERTISING SALES

Correspondence, letters and press releases should be sent to: Editor, **ST-LOG**, 9171 Wilshire Blvd., Suite 300, Beverly Hills, CA 90210.

Correspondence regarding subscriptions, including problems and changes of address, should be sent to: **ST-LOG,** P.O. Box 16928, North Hollywood, CA 91615 or call (818) 760-8983.

Correspondence concerning a regular column should be sent to our editorial address (see Authors below), with the name of the column included in the address. We cannot reply to all letters in these pages; so if you would like an answer, please enclose a self-addressed, stamped envelope.

J.E. Publishers Representatives — Los Angeles: (213) 467-2266.
6855 Santa Monica Blvd., Suite 200, Los Angeles, CA 90038.
San Francisco: (415) 864-3252. Chicago: (312) 445-2489.
Denver: (303) 595-4331.
New York: (212) 724-7767.

Address all advertising materials to: Paula Thornton — Advertising Director, **ST-LOG**, 9171 Wilshire Blvd., Suite 300, Beverly Hills, CA 90210.

PERMISSIONS

SUBSCRIPTIONS

ST-LOG, P.O. Box 16928, North Hollywood, CA 91615 or call (818) 760-8983. Payable in U.S. funds only. U.S.: $28.00-1 year; $52.00-2 years; $76.00-3 years. Foreign: add $7.00 per year per subscription. For disk subscriptions, see the cards at the back of this issue.

AUTHORS

When submitting articles and programs, both program listings and text should be provided in printed *and* magnetic form, if possible. Typed or printed text copy is mandatory and should be in upper- and lowercase, with double spacing. If a submission is to be returned, please send a self-addressed, stamped envelope with your manuscript to **ST-LOG**, P.O. Box 1413-MO, Manchester, CT 06040-1413.

COVER PHOTOGRAPHY:
GARRY BROD

COVER GRAPHICS:
DIGITAL ART

INTERIOR PHOTOGRAPHY:
STEVEN HUNT
KEE HWA JUEN
GARRY BROD

ILLUSTRATORS:
FABIENNE MASON
AWO

ST User

BY ARTHUR LEYENBERGER

I just received the latest edition of MichTron's *Griffin Gazette*. This newsletter is published quarterly and is packed with information on new products, hints and tips for MichTron's products and special deals for owners of their software. If you haven't seen it, you ought to take a look.

Granted, the *Gazette* is a vehicle for advertising MichTron's own products, but it goes beyond that. The particular issue I have in front of me is an excellent example of the type of quality support that ST software publishers should provide. This is a class act!

For example, owners of Microdeal's *Airball* will find some hints on how to find the Spell Book, the goal of the game. In addition, an enclosed coupon entitles you to purchase *Fleet Street Publisher* for $50 just by sending MichTron the copyright page of the manual from the DTP program you're currently using. If you are not yet using a DTP program, you can buy *Fleet Street* for 25% off the retail price.

As you may know, MichTron is now distributing *HiSoft BASIC* instead of *GFA BASIC*. If you already own *GFA BASIC*, you can purchase *HiSoft BASIC* or *HiSoft BASIC Professional* for half price, again by sending the copyright page of the GFA manual. The *Gazette* also includes information on other new MichTron products such as *ProText*, a fully integrated word processor; *Michtron BBS Version 3.0*; *Fleet Street Publisher 2.0*; *Hyperfont*, a GEM-based font editor; *HiSoft BASIC*; and *Grail*, a new adventure game.

If you are a registered user of any MichTron or Microdeal product, you should already be getting the *Gazette* free of charge. If not, give MichTron a call at (313) 334-5700. Some of the deals mentioned above may no longer exist by the time you read this. In any case, I commend MichTron for this effort and encourage other software publishers to follow their example.

Epyx power

Although it has been available for awhile, *Art & Film Director* from Epyx has been sitting in my pile of software to be looked at real soon now. I have finally had a chance to open the package and play with it for a considerable time. I am definitely impressed with this product!

Art & Film Director was originally created a couple of years ago and was to be sold by Broderbund Software. Broderbund never released the product and in the meantime, Epyx has picked up the rights and released it. *Art & Film Director* is really two products in one that allows you to create graphic screens which can then be animated.

Art Director is a full-featured paint program that provides you with the tools to produce 16-color works of art. Color cycling can be used to display up to 128 colors at once. Like other ST paint programs, on-screen menus and icons make it easy to select from a variety of geometric shapes or draw freehand. Free-hand painting can be done using "spray cans" or one of 40 pencil nibs in different sizes and shapes.

You can manipulate parts of your picture in a number of different ways. The Bulge feature lets you create either a concave or convex effect and wrap an image around a sphere. Spin will rotate a square or rectangle around either a vertical or horizontal axis. The Sprite feature lets you define a circle and bounce it around the screen.

Other features include shadowing for three-dimensional effects, using a portion of the picture as a paint brush or fill pattern and perspective. The program also makes creating or changing details easy with multiple levels of zoom and a "window" feature that allows a specific work area to be blocked and changed without affecting surrounding areas. Two separate screens can be worked on at once or combined in several ways.

Film Director lets you add the animation to bring your creations to life. Easy-to-use menu-based commands let you place characters, props, etc., on a frame and

then link the frames together to make a film. In addition, built-in music and sound effects are available to add the finishing touch to your "film."

The program uses cel animation to let you modify a portion of a frame without tediously redrawing the entire frame. Notable is the "tweening" feature that automates the process of animation right before your eyes. It's as simple as defining the starting and ending points—the program then automatically generates every image in between.

Art & Film Director comes with four disks (two for each of the two programs) and an excellent manual. The documentation includes a tutorial and a quick reference section for both of the programs. A nice touch is the information describing how to record your animation sequences on video.

Included in the package is a program to convert *DEGAS* and *Neochrome* files into *Art Director*'s .ART file format. Sample files are also provided that contain ready made art and animation sequences. These are especially helpful for learning how to harness the power of the program.

The powerful *Art & Film Director* package sells for $80 and is available from Epyx, 600 Galveston Drive, Redwood City, California 94063. They can be reached at (415) 368-3200. *Art & Film Director* is a top-notch product that really shows off what the ST can accomplish. If you yearn to be creative with your ST, I highly recommend this program.

Game fever

Over the years there have been a number of games that I have found particularly addicting. Titles such as *Missile Command*, *Space Invaders*, *Pac-man*, *Boulder Dash*, *Seven Cities of Gold*, *Time Bandits*, *Mean 18*, *Arkanoid* and *Tower Toppler* come to mind. To one degree or another, I have played all of these games for hours on end. They are challenging, fun and, yes, addicting.

A new game has recently captured my interest. So far, I estimate that I have logged over 100 hours playing it. It is *Tetris*, from Spectrum Holobyte, and is one of the most enjoyable games I have ever played on any computer.

Like many of the games mentioned above, *Tetris* is simple in concept. Basically, it's a game of eye-hand coordination. The goal is to rotate and position various-shaped blocks that fall from the top of the screen into a solid row at the bottom.

When a solid row is formed you are awarded points and it disappears. Gaps often are left in a row, especially at the higher game levels, which causes rows to build up line by line. The game ends when there is no more room for blocks to fall. Since the specific shapes appear in random sequence, strategic thinking is required and frequently the fate of a game rests on how you decide to play a particular block.

When the shapes appear at the top of the screen, you rotate and maneuver them with either the arrow keys or the J, K and L keys on the keyboard. The spacebar is used to drop the piece to the bottom of the screen once you have the piece in the right orientation and position. The faster you drop the pieces the more points you get.

After a set number of rows have been completed, the game moves to the next level (*Tetris* offers nine levels of play) where the pieces fall at a faster rate. You can begin the game at a higher level, either at the start or any time during the game. For additional challenge, you can start the game with up to seven randomly created rows already on the screen. More starting rows means a potentially higher score.

Each new screen has a different background graphic, including Mayday celebration at Red Square, Matinee at Bolshoi Theater, view of Earth from Solyut Space Station and game day at Lenin Stadium. Although the backgrounds are incidental to the actual game play, they are well-designed and detailed.

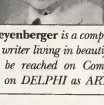
Sound effects can be turned on or off, statistics can be displayed on screen showing the number of each shape positioned and a help menu is available. Another feature allows you to display the next piece that will fall. Using this feature is mandatory if you want to get high scores. A high-score screen is also available which shows the top-ten comrades and scores.

Tetris was invented by a 30-year-old Soviet researcher named Alexi Paszitnov who currently works at the Computer Centre (Academy Soft) of the USSR Academy of Scientists in Moscow. The original programmer was 18-year-old Vagim Gerasimov, a student studying Computer Informatics at Moscow University. *Tetris* has been called the software equivalent of the Rubik's Cube. Even closet gamers will enjoy it, given the quality of the game and its simple, yet addicting, nature.

The ST version of *Tetris* sells for $40 and is available from Spectrum Holobyte, 2061 Challenger Drive, Alameda, California 94501. Call (415) 522-3584 for more information.

As much as I enjoy *Tetris*, it is a two-dimensional game. Blocks can be moved left and right, rotated and dropped. I would love to see (and play) a three-dimensional version. In addition to moving blocks left and right, a 3-D game would allow them to be moved forward and backward. Further, they could be rotated front-to-back as well as counter-clockwise.

Think of it: You would have to form not only a solid row but a complete set of rows on the bottom level before it would disappear and get points. Perhaps as each row is formed, you could position it anywhere in the front-to-back space. A 3-D *Tetris* would be *really* challenging. Are there any software authors out there willing to take on this challenge? ▪



Arthur Leyenberger *is a computer analyst and freelance writer living in beautiful New Jersey. He can be reached on CompuServe at 71266,46 or on DELPHI as ARTL.*

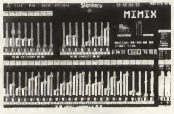
ST NEWS

FOR JULY 1989

Desktop Mixing

Steinberg/Jones, a company well known for their MIDI software, has just announced a new product that may be the next step into the future for the computer/music combination. *MIMIX* is a multichannel software-based mixing console that offers a screen display including 42 VCA levels, 34 VU meters and 180 various switches, all of which may be manipulated with the mouse.

Audio signals are displayed using a needle meter with an LED peak indicator or using a bar graph with a peak hold meter. Each channel on the mixer features a real-time VU meter; mute, solo and solo defeat switch; read/write update mode; VCA level; and a 24-character name.



The hardware portion of the system is the MIMIX VCA Module, which is contained in a 19-inch rack-mountable unit and includes a real-time noise gate with ten programmable parameters. The MIMIX system can handle eight of the VCA Modules.

The MIMIX system contains too many features to list here; anyone interested should contact Steinberg/Jones. But you'd better look at the prices first: $5,995 for the 16-channel system and a whopping $19,995 for the full 64-channel system.

Steinberg/Jones
17700 Raymer Street, Suite 1002
Northridge, CA 91325
(818) 993-4091
CIRCLE #130 ON READER SERVICE CARD.

Extra serial ports

db Technology has announced the availability of its dual-port serial interface for the ST computers. The SPIIST, which comes with standard DB25 connectors and data activity lights, offers baud rates of 50 to 38.4 kilobaud. With this unit comes the potential for multiuser bulletin boards, and future enhancements will expand the capabilities to allow local area networks for up to 255 users.

The retail price of the SPIIST is $129.95.

db Technology
P.O. Box 246
Cottondale, AL 35453
(205) 556-9020
CIRCLE #131 ON READER SERVICE CARD.

Dungeon helper

Dungeon Master has the honor of being the top-selling ST program of all time. People by the tens of thousands have ventured into its dank and dreary hallways, though not everyone who has sallied forth has managed to solve the many puzzles that block them from victory.

But help is available from many sources, not the least of which is Computer Publications, Unltd.'s *Dungeon Master Adventurer's Handbook*, a complete guide to this intriguing game. This concise 40-page volume includes complete descriptions of characters, magic spells, combat strategies and monsters, as well as thorough level-by-level overviews of the dungeons, including detailed maps. All objects and puzzles in the dungeons are referenced, with each puzzle's solution included for those who find themselves stuck.

The *Dungeon Master Adventurer's Handbook* is $8.95 and is available at the address below.

Computer Publications, Unltd.
P.O. Box 2224
Arvada, CO 80001
(303) 423-6805
CIRCLE #132 ON READER SERVICE CARD.

Mouse playground

Curtis Manufacturing Company has added the Curtis Mouse Pad to its product line. The pad is of laminate construction with a nonslip surface that the company claims helps maximize accuracy, control and response, as well as extend roller ball life.

According to Tom Judd, president of Curtis, "Anyone concerned with the accuracy, performance and longevity of their mouse needs a Curtis Mouse Pad. We've developed a cost-effective, quality product that delivers precise cursor positioning and improves mouse performance."

The Curtis Mouse Pad measures 8 × 9.5 inches and is waterproof and stain resistant. Its nonskid backing keeps it from slipping on your desk's surface. The pad retails for $6.95.

Curtis Manufacturing Company
30 Fitzgerald Drive
Jaffrey, NH 03452
(603) 532-4123
CIRCLE #133 ON READER SERVICE CARD.

Leisure Suit Larry is back

Just when everyone thought it was safe to play adventure games again, Sierra announced *Leisure Suit Larry Goes Looking For Love (In Several Wrong Places)*. If nothing else this game wins the coveted longest-software-title award.

This time around Larry has hit it big in the lottery and takes a romantic cruise, visiting several tropical resorts. But, as we all know, nothing ever goes right for poor Larry and spies from several nations make sure that this adventure doesn't change that streak of luck.

The game was programmed using Sierra's new development system, Sierra Creative Interpreter (SCI), which allows a full 320 X 200 graphics resolution and improved musical sound tracks.

The author, Al Lowe, who reportedly spent several months researching this new story, says, "I included in this game every resort and cruise I've enjoyed, so the IRS would let me write off my last three vacations."

The game was programmed using Sierra's new development system, Sierra Creative Interpreter (SCI), which allows a full 320 x 200 graphics resolution and improved musical sound tracks that can be routed to a Roland MT-32 sound generator or a Casio keyboard. Future Sierra adventures, including *Police Quest II* and *Space Quest III*, also will be developed with SCI.

Sierra On-line, Inc.
Coarsegold, CA 93614
(209) 683-4468

CIRCLE #134 ON READER SERVICE CARD.

More from Psygnosis

Psygnosis has just released an arcade/ strategy game for two players with the unlikely name of *Captain Fizz Meets the Blaster-Trons*. That's right, it's a game, not a science fiction serial from the '50s.

Captain Fizz Meets the Blaster-Trons is priced at $29.95

According to Psygnosis, if players have any intention of being successful in this quest, they must learn to play cooperatively. "The collaborative nature of the game is no mere surface glitter; you simply can't win without planning a strategy, watching your partner's back and even—when the chips are really down—sacrificing your own life so that your fellow player may go on to serve the noble cause. If you take the attitude that it's every man for himself, then the Blaster-Trons will finish you off in no time at all."

Captain Fizz Meets the Blaster-Trons is priced at $29.95.

Psygnosis
Distributed by
Computer Software Services
2150 Executive Drive
Addison, IL 60101
(312) 620-4444

CIRCLE #135 ON READER SERVICE CARD.

FROM OVER THE BIG WATER

By Marshal M. Rosenthal

It makes you want to tear your hair out—what little you might have left. Just a few years ago a reviewer was safe; overseas software *never* appeared in the States. You could wait months on end to mention a product, and still, few, if any, American ST users would have heard of it.

But now that's all changed. Now programs flow like quicksilver between Europe and our home—so quickly that it confronts us with a problem.

Remember how in the last column *Batman—The Caped Crusader* (Ocean Software) had just appeared in its small British box with two disks and a free poster that had the multilanguage directions on the back? You can bet that it's a hot topic right now, what with all the attention directed at "Pointy Ears" due to the major motion picture bearing his name—and this being his 50th anniversary. In the time since I wrote that last column, Data East has acquired distribution rights for *Batman* and released it, making it the review territory of other columnists. So, let's take a peek at *Afterburner* instead.

Activision U.K. has done a great job of capturing the feel of this mega-arcade hit from Sega: great music every step of the way, nifty sound effects and a digitized voice. You even get a poster and arm patch.

Afterburner places you behind the cockpit of a lean, mean F14. The two-disk boot-up runs some fancy graphics while you ponder options of music, sound effects and the sensitivity of the mouse control.

All controls are handled using the mouse—banking from left to right and diving or climbing, accelerating and firing (keyboard options are also available). To get the feel of the F14, try rolling through a 360° flip. This manuever will aid you in avoiding enemy aircraft and missiles approaching from behind—those that the flashing warning lights alerted you to.

The graphics are good; realistic enough to get the job done, while moving at a fast rate to keep the game from becoming boring (I would have liked a joystick option though). The background tune keeps the pace throughout each stage of the game, with sound effects having just the right *kerblam* and *whaboom*. Once off the deck of the aircraft carrier, make sure you don't run out of missiles or overheat your laser cannons.

It's worth noting for hand-held game fans that Tiger Electronics LCD version of *Afterburner* (complete with ¾-sized joystick and flashing lights) should be available by the time you read this.

Dragonscape (Software Horizons) continues the theme of arcade action. Fly your pet dragon, Garvin, through the realm of Tuvania, collecting and distributing vital artifacts along the way. Smoothly animated, Garvin can face in any of 16 directions. Five separate areas will scroll past you; the Wastelands, Woodlands, Icelands, Techno City and Arcadia. Each level is eight screens, with plenty of sur-

AFTERBURNER · Activision U.K.

AFTERBURNER · Activision U.K.

SUPER MENACE · Psygnosis

SUPER MENACE · Psygnosis

prises—such as the evil King of Chaos, who has dispatched his evil legions to hamper you (kill you, to be exact).

Twenty different enemies appear at each level, with a super baddie making his debut should you survive long enough. Meanwhile, traps have been laid to both delay and confound you, so just don't pick up everything that isn't nailed down! Yet somehow you must find and correctly place the eight key objects in order to end the Curse of Chaos. Did I mention the mazes surrounding some of the artifacts? Sorry.

On a more serious note, *Master Sound* (also Software Horizons) turns you on to the power of sampling at a low price (list is approximately $50). Features of this hardware/software package include variable automatic recording, filtering, compression of sounds and fading.

Sampled sounds may be sequenced, edited and assigned to one of 18 different presets, and then played back in real-

SPRITE MASTER · Soft Bits

SPRITE MASTER · Soft Bits

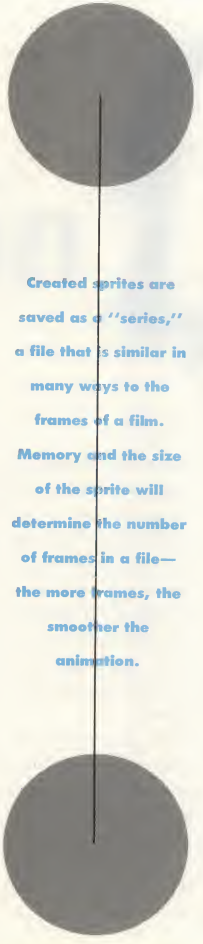
time. The package goes the distance by also including a utility screen that can display sounds along with user-controlled scrolling messages and animated accompaniment. Add to that a real-time VU meter, oscilloscope and 34-bar spectrum analyzer.

Now let's turn our attention to *Sprite Master* from Soft Bits. For those who may not know, sprites are those creations that move on screen: aliens, cute little guys, even explosions. *Sprite Master* lets you design your own sprites in low resolution, ranging from a tiny 16 x 16 pixel-sized figure on up to massive 144 x 84 version—and all using a full 16 colors. Created sprites are saved as a "series," a file that is similar in many ways to the frames of a film. Memory and the size of the sprite will determine the number of frames in a file—the more frames, the smoother the animation.

The editing screen is where sprites are born. Split into three sections, the left side is an enlarged view of the full sprite. Below to the right can be found the drawing/editing controls. These include many of the conventional drawing tools (pen, lines, box, fill, etc.), plus copy, flip, rotate, resize and palette control. Sprites are saved in memory with an individual frame number and color palette. Functions are de-selected easily with the right mouse button, and the finished file can be previewed and "stepped" through in a different screen. The speed of the frames can also be varied.

Soft Bits also includes *Picmaster*, a program that loads picture files and compresses them to half size so that you can cut and paste them for sprite use. *Listmaster*, also included, converts data from *Sprite Master* into an ASCII format to use in BASIC programs. A thorough technical information section brings up the rear of the book, and is appreciated. *Sprite Master* can't make you an artist, but it does provide the tools and elegant interface necessary for transferring those animation ideas into reality.

We conclude with a preview of two upcoming games from Psygnosis. *Super Menace* (tentative title) is indeed a super version of the popular *Menace* (and was written, in fact, by the same programmer). Those familiar with the game will find new excitement due to a horde of beasties, lots of color and both vertical and horizontal scrolling within the same level. In addition, you are not limited to a single type of spacecraft; depending on the planet you're heading for, you could be commanding a jetpack, helicopter or spaceship. It's shoot-em-up city, folks!



Created sprites are saved as a "series," a file that is similar in many ways to the frames of a film. Memory and the size of the sprite will determine the number of frames in a file—the more frames, the smoother the animation.

Blood Money, on the other hand, requires strategy as well as good reflexes—at least in the demo version I was viewing. You control a little guy moving through a horizontally scrolling world of magic and evil. Use the tools you find to bring joy and light to all—maybe you'll then be allowed to take a ride on the dirigible that keeps passing overhead.

See you next time.

***Marshal M. Rosenthal** is a New York-based photographer and writer specializing in children with product, video graphic enhancements and high-tech entertainment. His written/photographic projects have appeared in major publications in England, France, Germany, Sweden and the U.S.*

Products mentioned:

Afterburner
Activision U.K.
Blake House, Manor Farm Road
Reading, Berks
England RG2 OJN

Blood Money
Psygnosis, Ltd.
First floor
Port of Liverpool Building
Liverpool, England L3 1BY

Dragonscape
Software Horizons
5 Oakleigh Mews
London, England N20 9QH

Master Sound
Software Horizons
5 Oakleigh Mews
London, England N20 9QH

Sprite Master
Soft Bits Software
5 Langley Street
London, England WC2H 9JA

Super Menace
Psygnosis, Ltd.
First floor
Port of Liverpool Building
Liverpool, England L3 1BY

the Animati

BY MAURICE MOLYNEAUX

Over the previous four installments we've covered conceptualizing an idea, refining and storyboarding it. So you should have a fairly good idea of how the idea you want to animate will turn out. You have the "plot" and you have the "script." That's a lot, but not quite enough. Unless you're doing a flying logo or something mechanical, you probably plan to have a character or two (or more) appear in your animation. Since you've storyboarded the animation, it's possible that you've already designed any characters you'll need, but it isn't necessarily the case. Whether or not you have any character design done probably depends on how detailed your storyboarding and sketching has been up until now.

By the way, for the past two issues I used the ant and magnifying glass sequence to illustrate storyboarding, rather than using material from the music video I mentioned as my actual example project. I did this because it was simpler to illustrate the story and break down techniques with a sequence like that rather than trying to demonstrate it with the music video. The rough storyboards for that encompass over 70 drawings at the halfway mark! I'll return to the music video this time, but we'll keep our ant friend around for when we need him.

Defining the character

Even if you already have a good idea of how you want your character(s) to look, it's still a good idea to explore the possibilities to see if you can come up with something better. As with everything else in this process, the look of a character might give you some more ideas or suggest variations on existing ones.

The first thing you need to do is list all the characters that appear in your animation, and alongside their names specify character traits and all the details you know about them. You should try to form a mental picture of a character that fits the traits and the name.

If they don't have names yet, give them names. Oddly enough, you'll probably find it easier to "relate" to these fictional beings if you give them a handle. Try to find a name that fits the type of character you have in mind. For example, the little ant character seen in the previous two installments is a rather smart character. You don't picture him as big and brutish, so a name like Crusher or Ivan seems inappropriate because they are "strong" names. You want something a little smarter sounding. Then again, you don't want something too "brainy" sounding or you risk a name that implies nerd-dom. You need a good middle ground, but nothing

on stand:

Design for Living

ordinary. Nobody remembers ordinary names, in film or in real life. People pay attention to names like Atari, Symbolics, Pixar and Rhythm & Hues, but aren't interested in names like Software Industries or Computer Data Systems, Inc. The same goes with characters. A name like George White doesn't say anything, as opposed to interesting names that fit the characters who wear them. Below are a few from some animated films, with a pat character summation. Note how the names fit these descriptions.

Harry Canyon: Cab driver.
Elmer Fudd: A drip for all seasons.
Hop Low: Dancing Chinese mushroom.
Speedy Gonzales: Fastest mouse in Mexico.
Wile E. Coyote: Cunning carnivore.

See? Elmer Fudd is a dumb-sounding name, suited perfectly to the dolt who wears it. Hop Low sounds vaguely Chinese and implies movement and jumping, which the littlest dancing mushroom (from Fantasia) does. Harry Canyon is a pretty bland name, perfect for a grubby cabby in the 21st century; but it's unusual enough to be memorable.

A convention in older cartoons and comics was to give characters names that were alliterative (meaning starting with similar sounds) or rhymed. Think about it. Alliterative names include Clark Kent, Lois Lane, Bugs Bunny, Mickey Mouse, Daffy Duck, Atom Ant, Secret Squirrel. Rhyming names include Foghorn Leghorn, Spike and Tyke and Magilla Gorilla. There are even "punny" names like Chip and Dale and Mac and Tosh!

The song I am going to animate a video to is called "I Know You." The title, which is repeated numerous times in the song, is the inspiration for the animation. The line "I know you" implies recognition, and the song's verses reinforce this. Thus, the plot I came up with is a simple one: A little guy sees a good-looking lady, is instantly smitten, thinks she was made for him and absolutely won't leave her alone. The title is repeated six times every chorus, for a total of 36 times during the entire song. My image is of the lady trying to avoid the guy, and he pops up and points at her on every single "I know you." No matter where she runs, no matter what she does, he pops up—no matter how impossible or improbable the location. No matter where she turns, he's there.

The humor in the situation arises from his popping up from the most unexpected of places. To provide myself with a veritable playground of possibilities, the video is set in a museum, allowing the guy to appear with sculptures, in paintings, out of ash trays, doors, lamps, fossils—anywhere!

Within this framework you can see that the only kind of character that will work for this guy is an obsessive type, one who is oblivious to the fact that the object of his affections wants less than nothing to do with him. Also, he needs to be flamboyant, hammy. As the unwilling target of his affections, the lady merely needs to be aloof and disinterested, without being completely cold.

With regard to your own animation you should weigh the situations you've planned and find the key points to the characters there. Even if you've already defined your characters, it's useful to go through this step because you may find the personality type you've come up with doesn't work particularly well with the plot line you've established. For example, putting a meek little guy with easily bruised feelings into the video I have outlined simply would not work without some major plotting overhauls. Seeing a poor guy rejected wouldn't be as funny as

seeing an obnoxious guy get what he probably deserves (unless your gag material overcomes the inherent weakness of that character in the situation).

I have to repeat this: I am aiming for funny. *You* may not be and *should* not be unless that is specifically what you want and think is right for your project. Do what *you* want. My examples are just that: examples of the process.

Designing the character

Now that you understand the concept, I'll explain the character design. First of all, I knew right away the guy would be short and kind of unattractive and unfashionable—not ugly, but neither would he be tall or particularly handsome. The lady would, naturally, be a knockout; so he would have sufficient reason to go instantly ga-ga over her.

I wanted to get away from the classic cartoon style I'd used on Megabit Mouse. I wanted a kind of modern, angular look: very bold, with bright colors and hard lines—like a piece of modern art. The effect I was after was somewhat minimalist. Small details would be omitted. The only things that would appear would be what was necessary to show. If the guy's hands weren't doing anything important, they'd be simplified to mere shapes, void of details such as lines separating fingers. I pictured characters made up of geometric parts. Perhaps the guy would be made of round parts, the lady out of angular ones.

Round lost out quickly because it didn't look energetic or interesting. The music is up tempo; the pace of the planned action is fast. Therefore I felt the best character design would be something bold and hard edged, something that would look good even in fast movements. Following this thinking, a suggestion was made to give the guy a square-shouldered look, almost like a zoot suit. My first inclination was to give him a triangular body, rectangular limbs and wedge-shaped shoes. A sharply angled head didn't appeal to me, so I made his head and hands round. Figure 1 shows how I drew up a model of this concept on the ST to see what it would look like.

I got the opinions of several friends on this model. We agreed that while the general concept was on target, there was room for improvement. The round head didn't look right. Someone commented that the tails I'd indicated on the bottom of the suit triangle should be separated. Everyone thought the coloring appropriately tacky (this dude has no taste in clothes).

I whipped out a sketch pad and started scribbling. I doodled the guy doing some dynamic things, to get a better idea if this model would work. I could see how splitting the tails on the suit could improve the look of the character. I also played around with a geometric head shape, but hated it. I finally compromised with a rounded-off triangular head, which seemed to fit the bold design without being too harsh. I quickly drew a couple of these poses on the ST and realized I was getting closer. I began to think that the coloring on the guy's suit didn't clash enough; so I decided to try a check pattern made of green lines on the magenta suit and made the bow tie and shoes green also. As you can see in Figure 2, the suit pattern was always planned to be completely flat, with no effort taken to create an illusion of depth or to make the pattern "follow" the lines of the character's clothes.

How can I sum up this look? Perfect! Perfectly tacky!

Up to this point I had kept the guy's head blank—no ears, nose or mouth. The only face was made up of two horizontal lines for his eyes. This was in keeping with the minimalist ideal, but left his expressions rather flat. I experimented with his design a bit more, drawing him in a number of postures and positions. I found out that even the most energetic and emotional poses were weak without a real face. I tried modifying these sketches, adding appropriate cartoon eyes. I refrained from adding any other details (the mouth would appear only when necessary, like if he smiled sheepishly). The cartoony eyes suddenly opened up his range of expression, and he instantly gained an appeal the earlier designs had lacked. I tested these designs on the ST and the design was finalized as in Figure 3.

Well, more or less finalized. Small refinements will come when the first animation is tried. When you start moving the character, you begin to find design flaws, which can usually, but not always be easily corrected (legs may need lengthening, and so on).

The lady proved to be a more difficult proposition. I had to try to make a bunch of angular shapes look like a good-looking woman. Not easy! It took a lot of sketching before I got something that worked. As with the guy, it wasn't enough for this woman to just look good. She had to be flexible enough to go through the motions required, stylistically match the guy, a*nd* look good. I couldn't make her in a completely different fashion. I wanted graphic harmony, not discord!

While such stylized characters are in some ways easy to work with, they tend to have a rather limited scope. Their range of movement and expression is curtailed by their simplified design. A frown doesn't come across well without a mouth, and sniffing is tough without a nose.

A character like Megabit Mouse is another matter entirely. Like film cartoons of old, he is built up from round elements, which not only makes him cuter but also simplifies things because many of these building blocks maintain the same shape at any angle. To build shape tables of the guy in the music animation would require scores of different bodies in varying positions. Megabit's body is a sphere, which looks the same at any angle, so only one is needed (more if his size is to change). To give this ball-shaped body some direction a tummy spot is added (as a separate element), and his limbs are used to complete the illusion that there's more detail than actually exists.

The ant discussed previously could be drawn in a number of ways. As to basic body structure, because he's an insect a segmented body (head, thorax and abdomen) would be a good idea. The amount of detail, modeling, or lack of it, would permit dozens of wild variations even within this basic structure.

In the original rough storyboards, the ant had six limbs. By the revised boards (last issue) he had only four. The extra two got in the way, would have complicated the animation and weren't really necessary. They just complicated the matter without adding anything, so I left them off. Again, such decisions must be made on a case-by-case basis. If I'd storyboarded different actions I might have found the extra limbs useful and left them on. You'll have to decide what is desirable and what is better left out.

If you look at Figure 3 again, you'll notice that one of the things I was doing in drawing all these poses was to get an idea of the character's range of flexibility: what kinds of expressions and postures I could expect to get. It was a way of exploring the limits of the character's flexibility *before* committing a lot of time to an animation test. I could see if the parts were proportioned correctly, fit together in the right way. It also let me see if the character looked good and gave me an idea of the possible appearance of the final product.

There are a lot of different styles and approaches to character design; so many that to try to discuss even the most basic

forms would take more space than I have. Let's just say that styles in animation can vary as much as in any other graphic art. There are hundreds of different approaches you can take for any given character, and in a lot of cases no one of those is *the* right one.

I highly recommend generating test models of your characters on the ST as soon as you have a good idea of what they'll look like. You may find that the resolution and limited colors force you to make changes in your designs. This applies no matter which program you are using. If you're generating 3-D model characters, try building a rough model to see if it looks okay. If using a more conventional paint/animation system, draw it. It's also useful to try to determine your color palette at this point. Normally we've got only 16 colors to play with, and you should endeavor not to waste any. In the case of the man and woman in the music animation, I was careful to draw them with only eight colors. The remaining eight can be changed to suit the needs of a specific shot. To keep this consistent I created a master-palette screen that lists all 16 colors and indicates which are reserved and for what purpose.

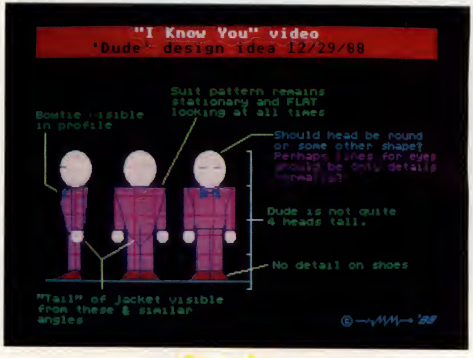


FIGURE 1

Endless pose-abilities

There's a dying art in animation known as character posing or just plain "posing." When you pose a character, you put the character in a position, posture and attitude that convey something about him/her or the situation they are in. For example, if you needed to scare a character, the first thing that most people would think of is drawing a wide open mouth and bulging eyes. All fine and good, but it's not as effective as combining that expression with body language, which can heighten the effect. When you're angry, you tense up. When you're sad, you tend to droop. People are very face-oriented, but if you look at most people, you'll notice that their body language can often be as revealing as the look they're wearing on their faces.

Using poses in animation is a somewhat tricky business. It requires you to find the pose that conveys the information you need, but at the same time will allow for a smooth transition into the next position (unless you *want* a complex, difficult transition). It must also fit in the scene. This may not sound all that difficult until you try it. One of the most difficult shots I ever had to storyboard was posing the guy in the music animation as he knocked on a door, opened it and

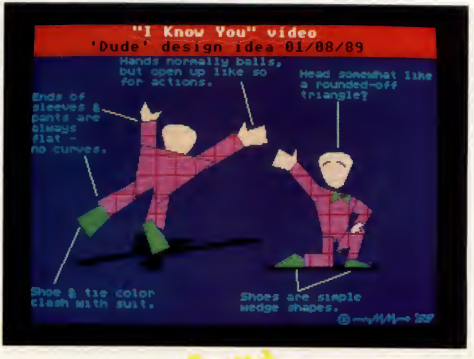


FIGURE 2

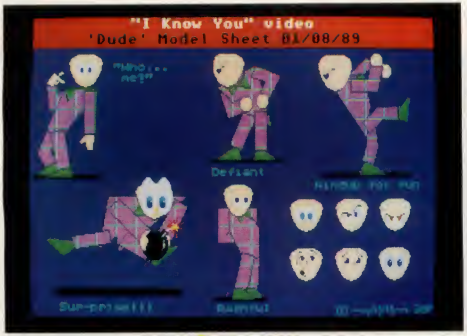


FIGURE 3

leaned through the door frame.

Probably the master of posing animated characters is Chuck Jones, director of hundreds of Warner Bros. cartoons. He described an animator as an "actor with a pencil," a statement which he could well apply to himself. His cartoons, particularly those dating from the late 1940s through the end of the 1950s, are beautifully staged, with expressive poses punctuating the situations and dialogue. The transitions between these poses are also excellent. You are not really aware that the poses are being struck, because they are integrated so well into the action. Refer to Figure 3 once again, and you'll see some relatively simple but effective posing. The character's whole body reflects each of the expressions and makes them clearer to the audience. This is of great importance considering the rapid pace of most animations. You have to find the best way to convey the information as quickly and clearly as possible.

Creating a series of poses and then going back and adding the movements between them is known as "pose to pose" animation. Animating a sequence from start to finish without using the pose to pose method is known as animating "straight ahead." The pose-to-pose technique allows careful planning and well-staged and choreographed action. The straight-ahead method is more spontaneous and sometimes results in more interesting output, though animating without a road map like this can also leave you with an unusable sequence.

If you have multiple programs, you should weigh the capabilities of each and decide which program will allow you to do the job with the least labor, and, more importantly, which program will produce the best results.

Due to space limitations there are a lot of subjects I can only touch upon. Posing is one of them. Some specifics related to posing within an animation, such as line of action and silhouetting action, we'll tackle when we get to the actual act of animation.

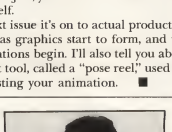
Which program?

We're at the juncture where, if you haven't made up your mind already, you need to decide what program or programs you'll be using to create your animation. If you have multiple programs, you should weigh the capabilities of each and decide which program will allow you to do the job with the least labor, and, more importantly, which program will produce the best results.

In the case of the music video, my plan is to animate the characters using *Film Director*, because it allows the greatest editing control and flexibility for cel animations. But, I'm not stopping there because speed is important as well, in addition to various "camera" moves planned into the video that *Film Director* cannot handle. Thus, once the characters are animated, the *Film* animation will (laboriously) be converted into a Delta file format and moved into *Cyber Paint* for polishing. *Cyber Paint*'s playback is faster than *Film*'s, plus it's video-effects-type tools will allow the addition of such effects as motion blur, image distortions, and so forth. Several other programs will also be involved, including *CAD-3D* and *Cyber Control* (to plot out a rotating mobile), and an old ray-tracing utility (why? I'm not telling yet!). On the hardware end, there's the Genlock I had installed in my ST last week (see *Step 1* in this issue).

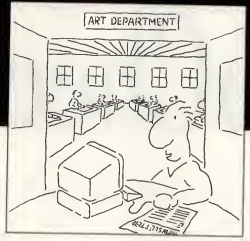
Why all these tools? As I said at the beginning of this series, I come up with the idea first, *then* pick the tools to do the job. The fact that I have to use a lot of them doesn't thrill me, but if there were an easier way to do what I want, believe me, I'd do it. There's nothing wrong with doing it the easy way—as long as the easy way is the *right* way. As usual, in regard to your own project, you'll have to decide that for yourself.

Next issue it's on to actual production stuff, as graphics start to form, and test animations begin. I'll also tell you about a neat tool, called a "pose reel," used for pretesting your animation. ■



Maurice Molyneaux *couldn't think of anything smug or impressive to say about himself this month—but don't expect it to last long.*

Open your own art department.



If you're a desktop publisher with big ideas and a small crew, let Migraph staff the desktop art department of your dreams.

Picture this: Professionally drawn images and illustrations at your fingertips. Powerful drawing tools, extensive editing tools, and a complete paint program at your command.

Plus high-quality printouts. Every time.

All that and compatibility, too. Migraph files load easily into your favorite Atari ST publishing programs.

Powerful. Versatile. And easy to use. Migraph's the ideal candidate for every job in your graphics department.

Touch-Up The complete design tool for high-resolution monochrome images.

Easy-Draw™ The professional object-oriented drawing program.

Supercharged Easy-Draw™ Easy-Draw plus basic publishing features.

Easy-Tools A 5-tool GEM desk accessory to enhance Easy-Draw.

DrawArt Professional A library of over 150 professional line art drawings.

Scan Art™ A library of over 100 high-resolution, bit-mapped images.

Border Pack A library of over 40 attention-getting border designs.

OSpooler A configurable background file spooler and print buffer.

Whatever desktop graphics project you have in mind—be it big or small, simple or ornate, traditional or avant-garde—Migraph's got you covered.

See your Atari ST® dealer today for more details.

MIGRAPH™ INC

200 S. 333rd St., Suite 220 Federal Way, WA 98003 800/223-3729 206/838-4677

CIRCLE #103 ON READER SERVICE CARD.

MicroCheck ST

BY CLAYTON WALNUM

It has been over four years since *ANALOG Computing* published the original version of my home-checking program, *MicroCheck*, and as amazing as it may seem, I still regularly get mail about it. It's not often that an author's first published work gets that kind of attention.

Of course, because of the original *MicroCheck*'s popularity, nothing would do but that I sit down and write a version for the ST. "It'll only take about six months," I said to myself.

Two years later. . . .

Note: Due to the large size of this program, it is available only on this month's disk version or from the databases of the ST-LOG ST user's group on DELPHI.

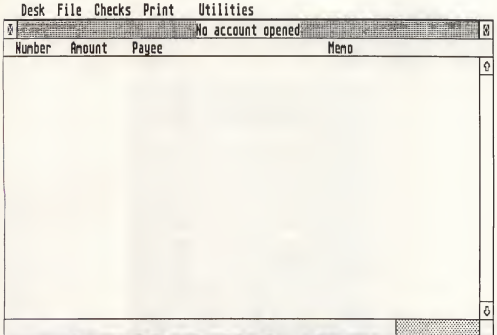


FIGURE 1

To run *MicroCheck ST* from the desktop, double-click on the file MICROCHK.PRG. (Make sure the file MICROCHK.RSC is in the same directory.) When the program has loaded, most of the screen will be filled with a window, and it is here that your check data will be displayed.

Getting started

To run *MicroCheck ST* from the desktop, double-click on the file MICROCHK.PRG. (Make sure the file MICROCHK.RSC is in the same directory.) When the program has loaded, most of the screen will be filled with a window (see Figure 1), and it is here that your check data will be displayed. Across the top of the screen is the menu bar. *MicroCheck ST*'s various functions can be accessed by selecting them from this menu or by pressing the equivalent keystroke commands. (The keystroke required is shown next to each item in the menu. Hold down Control and press the appropriate key.)

Across the bottom of the screen are six boxes that contain various information about your account. From left to right they are the account balance, the number of transactions, the number of checks and the number of deposits in the currently opened month, the month you're working on and the date. Before an account is opened, most of these boxes will contain zeroes.

The window contains scroll bars and arrows that will allow you to see information that doesn't fit in the window. You use these in the normal GEM fashion, clicking or dragging them with the mouse pointer. Scrolling the window to the right will allow you to see the dates on the checks. All other information fits in the window. Vertically, the window will hold 16 checks. If the window is fully opened by clicking on the full box in the upper right corner of the window, you can fit 20 checks, but you won't be able to see the information boxes at the bottom of the screen. If you have more than this number of checks in the current month's data, you can view them by moving the appropriate scroll bar or by clicking on the arrows.

Setting the date

MicroCheck ST uses the date shown in the date box for printing on reports. When the program is first run, it gets this date from the computer's system clock, so at the beginning of a *MicroCheck ST* session, you should make sure that the date shown in the date box is correct. If your ST has a battery-backed-up clock, or if you've already set the ST's date from the desktop, the date shown in the date box should be okay. Otherwise, you'll need to set it yourself.

To set the date, select the Date option from the Utilities drop-down menu or press Control-D on your keyboard. A small dialog box will appear. Enter the new date in the form mm/dd/yy, and then click on the OK button to install the new date.

If, after selecting the Date function, you decide not to change the date, you may cancel the operation by clicking on the dialog box's CANCEL button.

Starting a new account

The first thing you must do to use *MicroCheck ST* is create a new account. This procedure creates all of the files the program needs to keep track of your checking activity. You need to perform this process only once for each account you want to start.

Most of you will have only one account on your data disk; however, you can have as many accounts on your disk as will fit. Keep in mind, though, that as you add transactions to an account, its files will get larger. Make sure you have enough room on the disk. An average home checking account needs about 50K of data space, plus an additional 72K for the *MicroCheck ST* program and resource files.

To create an account, select New from the File drop-down menu or press Control-N. A dialog box will appear, prompting

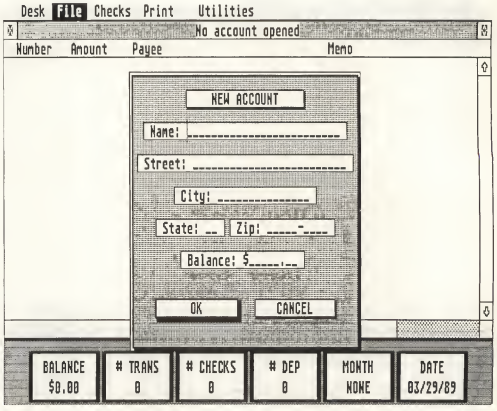


FIGURE 2

To create an account, select New from the File drop-down menu or press Control-N. A dialog box will appear, prompting you for the personal information the program requires. Fill in your name and address and the starting balance for your account.

you for the personal information the program requires. (See Figure 2.) Fill in your name and address (this information will appear on the check-entry dialog box, simulating the appearance of an actual check) and the starting balance for your account. Note that you don't have to enter a full nine-digit zip code; the program will be perfectly happy with only the first five digits. You can move between the dialog box's various fields using the up and down arrows on the keyboard. (The Tab key will move you forward one field.) When you've entered all the information properly, click on the OK button to proceed. At any time, you may click on the CANCEL button to discontinue the creation of a new account.

When you click on the OK button, another dialog box will appear, asking for the account's base filename. The program will use this filename as the starting point for creating all the files needed for your account. For example, I might want to name my account WALNUM. When I enter this filename, *MicroCheck ST* will create a file for each month, named WALNUM1.DAT, WALNUM2.DAT, WALNUM3.DAT, etc. The program will also create a file named WALNUM.MCK, which will contain the information that I entered in the new-account dialog box.

Enter your account's base filename (you're limited to six characters), and then click on the OK button to finalize your entry. If you wish to discontinue the new-account process, click on the CANCEL button.

When you click the OK button, the program will create your on-disk account. After this process is completed, the program will display yet another dialog box, prompting you for the month you wish to open. (See Figure 3.) Select the month by clicking on the appropriate button and then on OK.

Because your newly created account contains no data, an alert box will appear informing you that the current month file is empty. Click on the YES button if you want to start entering transactions into your new account. Click on the NO button to leave the new account as it is.

If you have been using the 8-bit version of *MicroCheck* (published in the February and March 1985 issues of *ANALOG Computing*), it is possible to port the data for your account over to the ST version. For information on how to do this, see the section titled "Porting 8-bit files."

Opening an account

The account creation process described above need only be done once for each account that you wish to use. However, every time you run *MicroCheck ST*, you must open the account that you want to work on.

To open an account, click on the Open selection of the File menu or press Control-O. A GEM file selector will appear. Double-click on the .MCK file for the account you wish to open, and you will then be presented with the month-selection dialog box. Click on the appropriate button; then click on OK to continue—or CANCEL to abort the Open function.

If the month you open contains no data, you will be asked if you wish to start a new month. Click on the YES button to open the month. If you click on YES, the new month will be opened, and any automatic transactions you have in your .AUT file will be added to the new month's data. (See the section titled "Automatic transactions" for more information.)

When a account is first opened, *MicroCheck ST* is in the edit mode as indicated in the window's title bar. (See Figure 4.) In the edit mode you may enter new transactions or modify previously entered transactions.

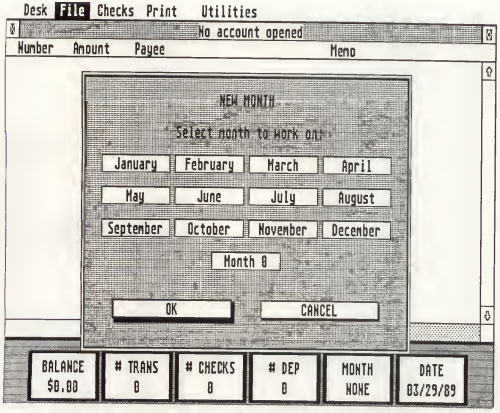


FIGURE 3

Enter your account's base filename (you're limited to six characters), and then click on the OK button to finalize your entry. When you click on the OK button, the program will create your on-disk account. After this process is completed, the program will display yet another dialog box, prompting you for the month you wish to open. Select the month by clicking on the appropriate button and then on OK.

Entering checks

Once you have an account opened, you'll want to begin entering checks. To do this, click on the Enter entry of the Checks menu or press Control-E. The check-entry form will appear. (See Figure 5.)

The check-entry form contains fields for all of your check's data, plus three buttons across the bottom. You can move between the check fields by using the up and down arrow keys. The tab key will also work, moving the cursor forward one field at a time. To jump quickly to a specific field, click on the field with the mouse.

The check number is, of course, the number of your check. This field will automatically advance by one each time you enter a check; so if you're processing your checks in order, you won't have to type anything in this field after the first check has been recorded. Note, however, that there are two reserved check numbers that you may not use for your normal checks: 0000 and 9999.

You should use a check number of 0000 for any transaction (other than a deposit) that was performed without a written check. For example, you might withdraw money from your checking account using an ATM (automatic teller machine). Even though you haven't actually written a check, you must nonetheless record this transaction.

The check number 9999 is reserved for deposits. Any time a transaction credits your account, this check number signals *MicroCheck ST* to add the amount of the transaction to your balance rather than subtracting it. Don't enter anything in the Payee field of a deposit (it won't hurt anything if you do, but you'll be wasting your time). *MicroCheck ST* automatically places the word "DEPOSIT" in this field when the checks are displayed in the check window. (Nothing will appear in the check-entry dialog box's payee field when you first enter the check.) If the credit comes from something other than an actual deposit (for instance, an interest payment), use the Memo field to note it.

When all of the information for a check has been entered, you must click one of the buttons along the bottom of the check form—or simply press Return to enter the current check and set up the form for the next. Clicking on the NEXT button has the same effect as pressing Return. Clicking on the DONE button enters the current check, then closes the check-entry form. Clicking on CANCEL closes the check-entry form without entering the current check.

Editing a check

If you find that you must edit a previously entered check, use the mouse to click on the check's entry in the window, and the check-edit form will appear. This form looks almost identical to the check-entry form. The only difference is that the NEXT button is no longer functional, and the DONE button is now the default (the button that will be selected if you press Return). When the form appears, make whatever changes you wish to the check's data, and then press Return or click on the form's DONE button. If you click on the CANCEL button, the check entry will remain unchanged, even if you changed some of the fields in the check-edit form.

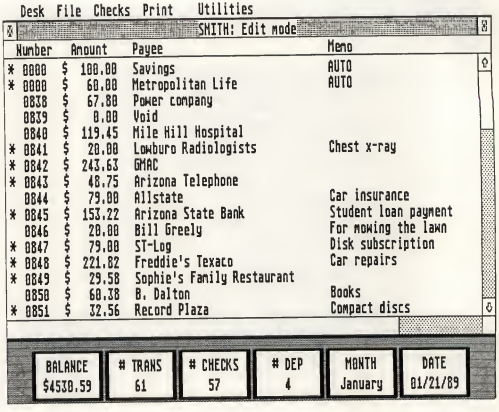
If you make any changes to the amount of the check, your balance will, of course, be updated to reflect those changes.

Searching checks

If you need to locate a check or group of checks, you can use *MicroCheck ST*'s search feature. To access this function, click on the "Search" option of the Checks drop-down menu or press Control-S. The search

FIGURE 4

When an account is first opened, *MicroCheck ST* is in the edit mode as indicated in the window's title bar. In the edit mode you may enter transactions or modify transactions that have been previously entered.



parameter-entry form will then appear. (See Figure 6.) When the form appears, each of the search parameter fields will contain default values. If you were to use all of the default values, you would be searching for every check in your account.

Fill in the parameters for the search, using the arrow keys, Tab key or mouse to move between the form's various fields. When you've filled in your search parameters, press Return to begin the search, or click on the OK button.

During a search, the parameters you entered are compared to the data found in each entry of your account. If the transaction matches every criterion in the search, it is added to the check list. Note that both of the text fields in the search parameters, Payee and Memo, will allow partial matches—that is, a payee search parameter of Ta will match checks with such payees as Tammy Brooks, Tabitha White and Tadbury Lumber. To find every check with a payee field beginning with G, just enter G as the payee parameter. Also note that the search function is not case-sensitive. To *MicroCheck ST* the letter "G" and the letter "g" are the same value.

A search is limited to a check list of no more than 1,000 entries. If the number of matches exceeds this (highly unlikely), an alert box will appear, informing you the search window will hold no more checks.

When the search is complete, a dialog box will appear showing the totals for the search. Click on the OK button or press Return to remove this dialog box, and the checks that matched your search criterion will appear in the check window. You are now in *MicroCheck ST*'s search mode, in which you may not edit or enter transactions. To exit the search mode and return to the edit mode, click on the Close entry of the File drop-down menu, press Control-C on your keyboard, or click on the window's "close box," located in the upper left corner of the window

Selecting a new month

Many times when entering transactions, you may have to move from one month to another. To close the current month and move to another, click on the New Month entry of the File menu or press Control-M. The new-month dialog box will then appear. To select a new month, click on the appropriate month button and then click on OK. To exit the dialog box without selecting a new month, click on the CANCEL button.

Printing check data

MicroCheck ST provides two methods of creating hard copies of your checking account. In the Print drop-down menu, you will find the selections Window and Register. The former is used to print only those checks in the current window; the latter prints the entire account. Click on the appropriate menu selection, or press Control-W or Control-G, respectively.

Cancelling checks

When you receive your statement from the bank each month, you must go through your account and mark those transactions that the bank has processed. This is the first step in "reconciling" your account, a process that ensures that your figures match those of the bank.

To enter *MicroCheck ST*'s cancel mode, click on the Cancel entry of the Checks drop-down window or press Control-P on your keyboard. A dialog box similar to the one shown for selecting a new month will appear. Click on the month you want to work on, and then select the OK button; the mode displayed in the window's title bar will then change from edit to cancel.

In the cancel mode, whenever you click on a check displayed in the window, in-

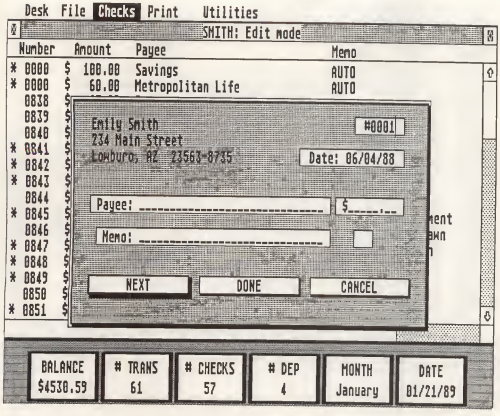


FIGURE 5

Once you have an account opened, you'll want to begin entering checks. To do this, click on the Enter entry of the Checks menu or press Control-E. The check-entry form will appear.

stead of bringing up the check-edit form, *MicroCheck ST* will place an asterisk next to the entry, indicating that the bank has processed that transaction. To uncancel an entry, click on it a second time. The asterisk will be removed.

When you're finished cancelling transactions, return to the edit mode by clicking on the Close selection of the File menu, press Control-C, or by clicking on the window's close box.

Reconciling your account

Once you've gone through your statement and cancelled all the appropriate transactions, you may reconcile your account. Click on the Reconcile entry of the Checks drop-down menu or press Control-R on your keyboard. A dialog box will appear, requesting your account's ending balance as shown on your bank statement. Once you've entered this amount, press Return, and *MicroCheck ST* will read through all your check files and, after making the appropriate calculations, display a final report. (See Figure 7.) If the amount shown at the bottom of this report is 0, then your figures agree with the bank's. Otherwise, either you or your bank made a mistake.

Press Return to exit from the reconcile-report dialog box.

Adjusting your balance

Sometimes, no matter how hard you try, you simply cannot get your account to agree with your bank statement. Usually this means that you've made a mistake somewhere in your account—maybe a check is made out for a different amount than what you recorded— that you have not been able to find. Although you should always try to reconcile your account properly, there may be times when you have to give up. In those cases, the best you can do is adjust your account's balance to agree with the one on your statement.

To perform this adjustment, enter a dummy check or deposit transaction for an amount that will bring your balance to the proper amount. When entering a dummy check, use the number 0000. These dummy transactions should be cancelled, just as if they had appeared on your statement.

Automatic transactions

Many banks can, at your request, set up your checking account so that it will automatically pay out checks at specified times each month. This type of service works out well for all the parties involved. It's convenient for you not to have to remember to write a check every month, and the payee (maybe an insurance company or a telecommunications service) can be reasonably sure it will get its payment regularly and on time.

Of course, if your checking files are going to be accurate, these transactions need to be entered into files just like any other. Fortunately, *MicroCheck ST* provides you with a method of handling automatic transactions, both debits and credits. You need enter this type of transaction only once, into a special file, after which time it will automatically be added to your account's data every time you start a new month.

To set up an automatic transaction, click on the Auto selection of the Checks drop-down menu or press Control-A. The check-entry form will appear with the memo field already filled in as AUTO. Simply fill out the check data as normal. When the form has been completed, and you press the OK button, the check will be added to your auto file (the file with the .AUT extension). Every time you open a new month (one whose file contains no check data), the transactions in the .AUT file will be au-

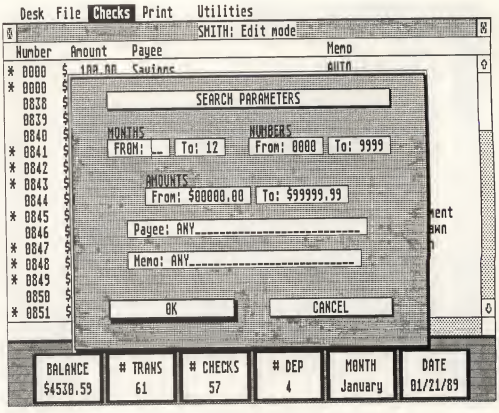


FIGURE 6

If you need to locate a check or group of checks, you can use *Microcheck ST*'s search feature. To access this function, click on the Search option of the Checks drop-down menu or press Control-S. The search parameter-entry form will appear.

tomatically added to that month's data.

If you've never entered an automatic transaction into your account, there won't be an .AUT file on your disk. When you attempt to add your first AUTO check to the account, you'll be asked if you want to start a new AUTO file. Click on the YES button, and the file will be created for you.

If you ever want to discontinue the use of the automatic transactions, simply delete the .AUT file from your data disk.

Closing your account

When you're ready to conclude a checking session, it's important that you close your account. *Failure to close your account properly may result in lost check data!*

To close your account, click on the Close entry of the File drop-down menu, press Control-C or click on the window's close button. You may close your account only when in the edit mode.

Quitting

You may return to the GEM desktop by selecting the Quit entry of the File drop-down menu or by pressing Control-Q. The Quit selection is a safe way of closing your account. All your data files will be properly updated before the program terminates.

Starting a new year

MicroCheck ST's calendar runs from January to December. That means that, come January 1, you must clear out the old files and start anew. *Before you do this make sure you have backed up your* MicroCheck ST *data disk!* If you fail to do this, you will have no way to refer to the previous year's files.

Before you can set up a new year, you must close the account you have open (if any). Then click on the New Year entry of the Utilities drop-down menu or press Control-Y. You will be asked if you really want to create a new year. If you click on the YES button, you will be asked again, just to make sure— remember, this function will destroy existing *MicroCheck ST* files on your data disk. Finally, a file-selector box will appear. Click on the .MCK file for the account you wish to set up for the new year.

MicroCheck ST will go through your entire account, deleting all cancelled transactions and moving uncancelled transactions into a special file for Month 0. The Month 0 file can be manipulated just like any other month's file and is simply a place where unprocessed transactions from the previous year can be stored until your bank statement shows that they have been processed. When you get your bank statement, you should go through Month 0, cancelling checks the same way you would for any other month.

When the New Year process has been completed, your account will contain check data only in Month 0 (and only then if you had uncancelled transactions in your account, which is likely); all other months will be cleared. (You made that back-up, right?) Automatic transactions, however, are unaffected by the New Year process and will still function properly with the new year's files.

Porting 8-bit *MicroCheck* files

For those of you who have been using the 8-bit version of *MicroCheck*, the ST version provides a function that will convert the old checking files for use with this program.

The first step in this process is to move the files from the 5¼-inch disk to a 3½-inch disk. There are two ways to do this. The first method requires a "null-modem" cable. A null-modem cable allows two computers running telecommunications programs to directly transfer files between them and may be purchased at most Atari computer dealers. To use the cable,

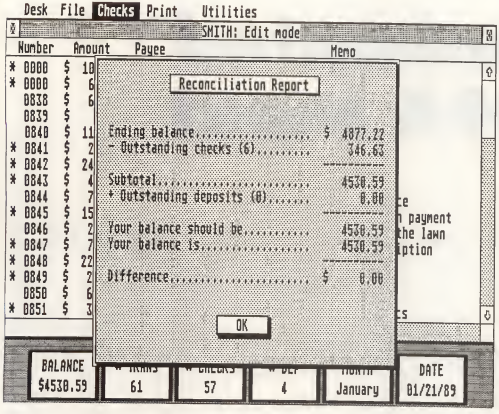


FIGURE 7

Click on the Reconcile entry of the Checks drop-down menu or press Control-R on your keyboard. A dialog box will appear, requesting your account's ending balance as shown on your bank statement. Once you've entered this amount, press Return, and *MicroCheck ST* will read through all your check files and, after making the appropriate calculations, display a final report.

connect one end to your ST's modem and the other end to your 8-bit computer's modem. Then run a telecommunications program on each computer. Set your ST to receive X-Modem, then do an X-Modem send of all the *MicroCheck* monthly files from your 8-bit computer.

Another way to do the transfer is to use your 8-bit computer to upload the *MicroCheck* data files to a BBS system or into your workspace on a commercial online system such as DELPHI, GEnie or CompuServe, and then download your files onto your ST. (If you use a BBS for this transfer, you'll need to make special arrangements with the SYSOPs. You don't want to find all your checking files in a public download area!)

Whichever method you use, make sure that you make the X-Modem transfer in the binary mode rather than in the text mode. And note that the only files that you need to transfer are the monthly data files. On your 5¼-inch disk, these files are named MONTH00.DAT, MONTH01.DAT, MONTH02.DAT, and so on. Further, note that empty months need not be transferred. When you run *MicroCheck ST*'s Import function, it'll create new month files for any 8-bit data files that are missing.

Once you have the files transferred, place them in the same directory as the *MicroCheck ST* program, and then run *MicroCheck ST*. Create a new account as described in the section "Starting a new account" above. For the account's balance you should use the balance shown in your 8-bit *MicroCheck* account. When the new account is ready, click on the "Import" selection of the Utilities drop-down menu. An alert box will appear, asking if your 8-bit *MicroCheck* files are in the same directory as the *MicroCheck ST* program. If you're ready to continue, click on the YES button or press Return. A file-selector box will then appear. Select the .MCK file for the new account you just created, and *MicroCheck ST* will then transfer your 8-bit checking data into that account.

If you had any automatic transactions in your 8-bit account, you will need to re-enter them using the method described in the section "Automatic transactions" above.

Conclusion

This program has been in the works—off and on—for over two years, and I have to admit that it is with great relief that I release it to the pages of ST-LOG. Although I suppose it's too much to hope that *MicroCheck ST* will be as popular as its predecessor was (according to some unwritten law of the universe, an author usually is allowed only one mega-hit per lifetime), I'm confident that you will find it a useful addition to your applications software library. Use it in good health—and may your account always balance. ▪

CLAYTON WALNUM

The author would like to thank Jim Gross for his relentless efforts in beta-testing this program and for his helpful suggestions. Jim's uncanny ability to crash even the most solid of programs has made this the best it can be.

Animated GFA Input

BY ALBERT BAGGETTA

In the movie *Short Circuit*, we hear the robot, No. 5, frantically calling for more input, a word very important to us programmers because the user has to have some way of communicating with the machine. No. 5 was able to communicate through voice input, along with a myriad of sensory inputs (sophisticated, indeed), but our home computers are still a long way from behaving in this manner, so we input mainly through the keyboard.

I'm not knocking the mouse, mind you, because I think it is a fast way of communicating with the machine on a symbolic level—click on the icon, and presto! You get a reaction. But when it comes to entering long strings of text into our home computers, the keyboard is still the most expedient peripheral.

I program most of my games for the Atari ST in GFA BASIC because its interpreter is easy to use and the source code can be compiled to a very fast .PRG file. GFA has a good selection of input routines, but I found that there was something else I needed. The built-in input procedures of GFA work in one of the following ways: either you enter a complete string or a single character. While you enter the string, all other activity of the program freezes. Hit Return and the program resumes its activities—useful, but boring, especially for the game programmer.

Single-character input can allow screen activity to continue but restricts the amount of input allowed. The program can be made to check for a certain key press, but once that key is pressed the program is back about its business. One character is not usually enough, unless you're working off a menu.

Suppose you want to enter a string of characters—a person's name, a guess at a question, an item for a list—and still want some activity on the screen simultaneously—some sort of animation, for example. It *is* possible and not very difficult to accomplish if you do some planning and use some imagination.

I was confronted with just such a dilemma while writing a game program recently. I wanted the user to be able to enter a response to a question, but at the same time I wanted a pulsating border around the input area. The accompanying routines show some of the code I used to accomplish this, and it can easily be adapted for other uses.

Listing 1 is a low resolution, demonstration GFA program I have written to show how you may enter some short input (as described above) while screen activity continues. When you run the program, you will see a border of asterisks surrounding the prompt "Enter Your Name:" Type your name (no more than 15 characters are allowed here), and then press Return to have the name string printed below the border. You will notice that while you are typing your name, the border animation continues without interruption.

If you study Listing 1, you will see that the first part of the program creates the animated sequence while the computer checks for the string value entered into *X$*. The *Inkey$* routine of GFA is used for this fast check. The source code section marked *INPUT EXAMINATION* is used to handle the input. First we check to see if the key pressed is "legal" before we show its character on the screen. (Everything is legal at this point except the Return key and the Backspace key. These will not print; they will result in an action. All other keys will be accepted and printed on the screen.)

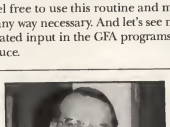
One of the problems I had with this routine was allowing the correction of errors. Out of frustration, I resorted to the easiest method: instead of using Backspace to correct one character at a time, I have it erase the entire line, making the user start over again. This is not the best way to handle the editing, but since the required input is short, no great burden is placed on the user.

The whole input sequence and animation is placed within a *REPEAT/UNTIL* loop. When *X$* becomes equal to a Return, or you have used up the allotted 15 characters, the routine ends, and your string is printed. In an actual program, you might print the resulting string, but more likely the string (called *Name$*, here) will be used to check for a correct answer or possibly might be used as part of a list.

The animation I created was done with the normal ASCII characters, but any kind of graphic or character animation can be incorporated in the loop. Because of GFA's tremendous speed, it takes quite a bit of screen activity before any hesitation appears in the actual input or animation.

I have put a delay loop at the end of the program, which is used during animation to slow down the speed of the flashing asterisk. If you play with the value in the delay loop, you can see how the movement is affected on screen.

Feel free to use this routine and modify it any way necessary. And let's see more animated input in the GFA programs you produce.



***Albert Baggetta** is an English teacher and a professional guitarist. He lives in Agawam, Massachusetts, with his wife, Beverly, and his two children. He frequently can be found wandering the* ST-LOG *SIG on DELPHI.*

Listing 1:
GFA BASIC 2.0

```
' ********************************************
' *                                          *
' *          Animated Input GFA              *
' *          by Albert Baggetta              *
' *                                          *
' *       Copyright 1989 by ST-LOG           *
' *                                          *
' ********************************************
'
' Set screen colors
'
Setcolor 0,3,4,5
Setcolor 15,7,7,7
'
' Set the starting points for animated box
'
Cd_rightX=5
Rd_downX=5
CX=10
RX=10
'
' Draw box of asterisks
'
Print At(Cd_rightX,Rd_downX);"*****************************"
For Ln_rowX=6 To 15
  Print At(Cd_rightX,Ln_rowX);"*                            *"
Next Ln_rowX
Print At(Cd_rightX,15);"*****************************"
Print At(11,7);"Enter Your Name:"
'
' Begin the main loop for animation and input
'
Repeat
  X$=Inkey$
  '
  ' Move the asterisk to the right on screen
  '
  If X$="" And Rd_downX=5 And Cd_rightX<32 Then
    Inc Cd_rightX
    Print At(Cd_rightX,Rd_downX);"*";
    @Del_ay
    Print At(Cd_rightX,Rd_downX);" "
    @Del_ay
    Print At(Cd_rightX,Rd_downX);"*"
    @Del_ay
  Endif
  '
  ' Move the asterisk down on screen
  '
  If X$="" And Cd_rightX=32 And Rd_downX<15 Then
    Inc Rd_downX
    Print At(Cd_rightX,Rd_downX);"*";
    @Del_ay
    Print At(Cd_rightX,Rd_downX);" "
    @Del_ay
    Print At(Cd_rightX,Rd_downX);"*"
    @Del_ay
  Endif
  '
  ' Move the asterisk to the left on screen
  '
  If X$="" And Rd_downX=15 And Cd_rightX>5 Then
    Dec Cd_rightX
    Print At(Cd_rightX,Rd_downX);"*";
    @Del_ay
    Print At(Cd_rightX,Rd_downX);" "
    @Del_ay
    Print At(Cd_rightX,Rd_downX);"*"
    @Del_ay
  Endif
  '
  ' Move the asterisk up on screen
  '
  If X$="" And Cd_rightX=5 And Rd_downX>5 Then
    Dec Rd_downX
    Print At(Cd_rightX,Rd_downX);"*";
    @Del_ay
    Print At(Cd_rightX,Rd_downX);" "
    @Del_ay
    Print At(Cd_rightX,Rd_downX);"*"
    @Del_ay
  Endif
  '
```

```
' INPUT EXAMINATION
'
'
' If a legal key is pressed, print it.
'
If X$<>"" And X$<>Chr$(8) Then
  Cx=Cx+1
  Print At(Cx,Rx);X$
  Name$=Name$+X$
Endif
'
' If the (backspace) key is pressed, erase the string so far
'
If X$=Chr$(8) Then
  For Ers=10 To Cx
    Print At(Ers,Rx);" "
  Next Ers
  Cx=10
  Name$=""
Endif
'
' If the (return) or the max length of string, jump out of loop
'
Until X$=Chr$(13) Or Cx=25   ! Max happens to be 15 characters here.
'
' Print the string -- here it is called Name$.
'
Print At(11,10);Name$
Print At(13,20);"Press A Key"
'
' Hold on screen for this demo until a key is pressed
'
Repeat
Until Inkey$<>""
Setcolor 0,7,7,7
Setcolor 15,0,0,0
'
' A delay loop to keep the asterisks from moving too fast.
' Make this a smaller number and watch the speed.
'
Procedure Del_ay
  For Dx=1 To 100
  Next Dx
Return
```

■ END

Attention Programmers!

ST-LOG Magazine is interested in programs, articles, and software review submissions dealing with the Atari home computers. If you feel that you can write as well as you can program, then submit those articles and reviews that have been floating around in your head, awaiting publication. This is your opportunity to share your knowledge with the growing family of Atari computer owners.

All submissions for publication, both program listings and text, should be provided in printed and magnetic form. Typed or printed copy of text is mandatory and should be in upper and lowercase with double spacing. By submitting articles to **ST-LOG Magazine**, authors acknowledge that such materials, upon acceptance for publication, become the exclusive property of **ST-LOG Magazine**. If not accepted for publication, the articles and/or programs will remain the property of the author. If submissions are to be returned, please supply a self-addressed, stamped envelope. All submissions of any kind must be accompanied by the author's full address and telephone number.

Send your programs and articles to:

ST-LOG Magazine
P.O. Box 1413-M.O.
Manchester, CT 06040-1413

STEP 1:

BY MAURICE MOLYNEAUX

Making Mixed Mirages Work

Graphics. These days it's next to impossible to work with a computer and *not* see them. Databases store pictures along with other data, word processors integrate text and graphics, and desktop publishing software goes even further. Heck, even the letters on your ST's screen are bit-mapped graphics, not a dedicated text-only video output. And if you play games or use a drawing program, graphics are even more unavoidable.

The graphics capabilities of personal computers have grown by leaps and bounds in the industry's brief 12 years of existence. Ever look at the "high res" graphics of an old Apple II? An incredible 280 x 192 pixels (a pixel is one computer generated dot on the screen) using eight colors! Actually, it took two of those horizontal dots to make one colored pixel. There were eight colors, but separated into two distinct, four-color palettes that were incompatible with one another, meaning you couldn't put a color from Palette 1 alongside a color from Palette 2 without color bleeding—*unless* the junction between those pixels fell exactly on a byte boundary! To further complicate this mess, each palette contained a black and a white color—so you actually got six colors, not eight!

The Atari 8-bits expanded this somewhat, adding numerous "modes" of different sized pixels with varying numbers of colors. Palettes ranged from a total possible 128 colors on early machines (eight luminances of 16 colors) to 256 colors (16 luminances of 16 colors). Although the normal modes limited the number of colors that could be used on a given screen (usually to four, although some GTIA modes allowed 16), with special programming every color in the palette could be viewed on a single picture.

In more recent years things have gotten better. The low resolution mode of the ST has more pixels on the screen than the highest resolution on the Atari 8-bit computers, a range of 512 possible colors, and normally allows 16 colors at once (and with special programming it's possible to put all 512 colors on screen at the same time). Medium resolution goes to 640 x 200 and four colors, and high resolution is 640 x 400 in pure black and white monochrome. Quite a jump from the Apple II! New monitors and graphics hardware soon promise modes upward of 1000 x 900 pixels on upcoming Atari machines.

So, as the hardware gets more powerful, the displays get better and better. Compare a *Spectrum 512* picture on the ST to even the best screen on an 8-bit Atari. Quite a difference. This upward spiral in graphics capability allows images created on a computer to become ever more realistic. More colors allow more nuances in detail and shading. Higher resolution means less noticeable pixels, making the picture look less "computerized."

In fact, the whole trend in computer graphics increasingly has been to "make it look real." This month the topic of *Step 1* will be hardware, software and graphics techniques that can make the ST's video output look its best.

Hues there?

Higher resolution makes smaller details possible, but what can you do when you've hit the limits of your hardware? The 320 x 200 pixel resolution and 16 colors out of 512 in ST low resolution are wonderful when compared to the graphics of the older computers described earlier, but then again those low-resolution graphics aren't so hot when compared to the output of most graphics workstations. Computer paint-box systems for video purposes (TV logos, commercials and so on) average about 712 x 480 pixels, with 256 colors out of a palette of 16 million! You don't see "jaggies" (the stair-step effect of pixels most clearly seen in diagonal lines drawn by a computer) on a TV commercial.

But resolution and total numbers of colors aren't the whole story. I've seen pictures on the ST that you wouldn't think were only 320 x 200 pixels. One great factor in computer graphics is how well you can disguise the weaknesses of your display.

Do you ever watch MTV? One of the regular spots that appear at commercial breaks is a series of short computer animations titled "Animals." Clean, impressive graphics (and funny, too, though that's not relevant here). I videotaped a lot of these, and one, titled "That Was a Wolf's Life," ended with a lightning bolt zapping the wolf/camera (it's always from the point of view of the title animal). I was curious about how the flashing effect was achieved, so I studied the lightning frame by frame. The effect was a simple white on black drawing (alternated with black on white every third frame or so to create a strobing pulse). But with the image so reduced down to two colors I was surprised by how clearly I could see the jaggies. The resolution isn't all that high. It's just that the large range of colors is used to blur the hard edges.

You can see this done on personal computers all the time these days (though not always as often as it could and should be), through a technique called "anti-aliasing."

This means, in short, to plug a pixel of a medium tone between two strongly contrasting colors. If you drew a dark red triangle on a bright blue background, the jaggies would stick out like, well, like jaggies. However, if at each obvious jaggie you inserted a pixel of a medium purple color, it would soften the junction and make the stair-step effect less obvious.

This can work well with as few as 16 colors if you choose your palette carefully. The trouble is manually doing this type of work is a real headache. There is some software that can help. Tom Hudson wrote an antialiaser desk accessory (available from Tom himself), which will work on screens in *DEGAS Elite* or *CAD-3D 2.0*. *Cyber Paint* also features its own antialiaser function, as does *Spectrum 512*.

To really get good output, though, you *need* more colors. A few programs provide this: *Spectrum 512* and *Quantum Paint* both boost the total number of available colors (*Spectrum* to the full 512 colors that the ST hardware can generate and *Quantum Paint* to a pseudo-4096—this done through pixel interlacing, meaning rapidly switching a pixel between one shade and another to create an effect of a hue in between.) With such programs it is relatively simple to kill the jaggies.

Getting better images onto your ST can also mean culling them from outside sources. One such source can be another make or model of computer. With the proper software a number of different graphics formats can be imported (and also exported). For example, *DEGAS Elite* can load Atari 8-bit "Koala" pictures (Graphics 7½ mode) and Amiga IFF picture files and then save them in *DEGAS* format.

The shareware *Pic Switch 0.7* by John Brochu can convert not only those formats, but also Atari 8-bit *Micropainter* and Graphics 8 and 9, *MacPaint*, and CompuServe RLE (run line encoded) graphics. Another program called *The C-64 Graphics Converter* (public domain) by Jerry L. Bethel (BETHEL on DELPHI) will import *Koala* and *Doodle* format pictures from the Commodore 64. There are also programs for converting to and from any IFF format, as well as the GIF standard (used in the IBM world).

Although many of these programs will convert these outside images to a standard ST low or medium resolution screen, a few will allow you to import pictures directly into *Spectrum 512* format, thus maintaining the appearance of a many hued picture better than whittling it down to four or 16 colors.

Smile and say "digitizer"

Drawing on a computer is rarely easy. Even given the best software tools and good input hardware (like a graphics tablet), it's still a pain compared to working in more traditional mediums. Admittedly, the drawbacks are slight in comparison to the gains. In what traditional artistic medium can you instantly change every instance of one color to another, clip portions of two images and seamlessly merge them together? Only with a computer.

But still, inputting the raw image to manipulate is rarely easy. One solution many have taken is to use a scanner or a video digitizer. These devices are used to break down images from outside sources and convert them into data the computer can display and manipulate.

The whole trend in computer graphics increasingly has been to "make it look real." This month the topic of *Step 1* will be hardware, software and graphics techniques that can make the ST's video output look its best.

Both scanners and digitizers work in roughly the same fashion; it's the manner of input that differs.

A scanner is usually a device with a small sensor that is passed horizontally over an image (usually a photograph or printed picture). Each time it passes, it scans the light reflected from the source image and breaks that down into numerical data. When that pass is completed, it then scans the next line. Software then assembles all these "line passes" into a single image, which can then be saved to disk and/or manipulated.

The human hand is too imprecise an instrument for this task so the common solution is to attach the scanner to a printer, usually a dot-matrix. The scanner's "eye" is mounted on or in place of the printer's print head, and the image to be scanned is fed into the printer like a blank piece of paper would be. The controlling software then makes the printer move its print-head positioning mechanism to move the scanner across one line of the image. When it has reached the end of that line, it lowers the print head and repeats the process. Really, it's sort of like inverse printing, using the printer as an *input* device.

Scanners are less often used for inputting images for display on the computer's screen than for creating data for use in desktop-publishing software and other printing utilities.

A video digitizer works slightly differently. The input is usually a standard composite video line, carrying the output from a video camera, VCR, video laser disk, television or even another computer. Like the scanner, the digitizer breaks this image down line by line—actually, video scan line by scan line. Because of this, the incoming image must be perfectly still. Therefore, either your subject must hold still for the required time, or, if you are using a videotape deck, you must be able to get a clear and stable (no flutter!) freeze-frame.

The average video digitizer for the ST takes between ten and 25 seconds to scan a picture and convert it into a ST-usable image. I'm talking about 16-shade or color images; the less tones and colors, the faster and "rougher" the scanning. There are digitizers which are known as "frame grabbers" that can "snapshot" a video image in real-time and turn it into a computer graphic nearly instantaneously. I am not aware of a digitizer for the ST that can do this, and if there is one, it is no doubt expensive.

The hardware isn't the only factor with these devices. the software that controls them can often be a major element. A number of good digitizers and scanners have been marred by buggy or inadequate software. When shopping for one of these, don't rush out and buy the first one you see or one that sounds great in an ad. Get recommendations from people who've used them. Try to see some of the actual output and find out how hard it was to get that output. I've seen some great digitized images that were nearly impossible to reproduce using the same hardware/software combination. Remember, demos are designed to look the best they can, often at great expense and time. You don't want to purchase a device that requires hours of grueling work to get one great image, when another one would give you similar results in minutes.

How many people have bought scan-

ners and digitizers in order to get realistic-looking pictures and images onto their computers? I'd say a lot, judging by the numbers of digitized pictures I've seen floating around. Admittedly, a lot are converted over from other computers. The lure is strong. You can capture and then edit images. The picture of myself that is now printed with this column is a digitized and retouched image. A live image of Megabit Mouse was digitized and then the cartoon character—me—was drawn in. One neat thing about digitized images is that you can exchange them with other people who have computers with compatible graphics formats like the aforementioned GIF, IFF and so on. I have seen the faces of a number of ST-LOG associates and fellow DELPHI members only by means of digitized pictures we've exchanged over modems and by way of disks.

I have one of each of these devices (the ComputerEyes video digitizer graciously provided by the fine folks at Digital Vision), and although I rarely use them for making digital snapshots, I do use them whenever I need a realistic element in a picture or animation. For example, I digitized some pictures of my hand for a scene at the end of the *Art & Film Director* video where a hand catches a disk. I would have preferred to use an actual video of a real hand, rather than a digitized image, but there was no way to do that on an ST until *now.*

TV on your ST

A little over a year ago I wanted to animate a scene of Megabit Mouse hopping out of the ST screen and landing on its keyboard. I didn't want him hopping onto a digitized picture of the computer, but onto an actual video image of it. I had no way to do that at the time, and the shot was abandoned. However, I could now do it rather easily. In fact, with the flick of a few switches I can be typing this text right on top of MTV. These words I am typing are (at the moment I am writing them) appearing over the faces of some obscure band's rock video! A twist of a dial and the words disappear.

How can I do this? Simple, I have a Genlock installed in my Mega ST. "Good for you, but what's that mean?" I hear many of you cry. A Genlock is a device that combines computer graphics with another video signal. More to the point, it places the computer image on top of the other video signal. With this device I can mix any low or medium resolution image with standard video. I'll have Megabit Mouse walking over the titles to *Star Trek IV,* then I'll place a *CAD-3D* generated starship over CNN's Crossfire, and maybe even whip out a paint program and draw a mustache and glasses on Dan Rather! All this and more is possible with a Genlock.

A lot of us have waited for a long time for someone to produce a Genlock for the ST computers. As I write this, only one company makes such a device, and, unfortunately, the current model is not for everyone (or even most of us).

The Genlock I have is called the JRI Genlock, produced by John Russell Innovations (JRI). It is an add-on card that hooks into the ST's video-chip socket and grabs the data for the ST's screen display. In the past I mentioned that this device would hook into the processor bus on the Megas and/or the DMA connector. I was mistaken. This card terminates in a small box, external to the computer, which features six connectors. One is a composite video input (for the image to be Genlocked upon), another is composite output (Genlocked image out), image out), in addition to a composite audio out (to output the ST's sound; there is no audio input. The standard ST monitor jack is carried through another port. The actual port of the ST itself is tapped into by the Genlock and inaccessible, hence this connector, and along with it is a port which outputs the Genlocked signal to an ST RGB monitor, meaning you can view video on your SC1224! The final port is a connector for the Genlock's supplied "remote control."

A lot of us have waited a long time for someone to produce a Genlock for the ST computers. As I write this, only one company makes such a device, and, unfortunately, the current model is not for everyone.

On the side of the box are three recessed controls that can be adjusted with a screwdriver. One adjusts the hue like the tint control on many color TVs and composite color monitors, another the horizontal position of the computer image, and the third allows the Genlock to "lock" to the input video signal under changing thermal conditions.

Installation is not easy and requires disassembly of the ST. The case and RF shielding has to be removed, the internal power supply disconnected, and the ST's "Shifter" chip has to be removed. The Shifter is plugged into a socket on the Genlock board, which is itself plugged down into the Shifter's socket on the ST motherboard. The Genlock is hooked to the power supply, and special RF shielding must also be installed, in addition to reassembling the computer and attaching the small box with all the ports. The manual that comes with the Genlock explains this installation, but JRI prefers a trained technician to do it, or to handle the job themselves, because, apparently, it is easy to damage the board: The parts are extremely static-sensitive. JRI will install it, if you want, but you'll have to ship your computer to them.

Be warned, this device is not cheap. It's not overpriced either. However, the original planned price of $500 had to be abandoned, John Russell told me, because at that price he would not have made any money unless every board produced worked perfectly and there were no defects. Perfection is extremely rare, and to cover repair and replacement costs and labor, the price had to go up. The current list price is $650, and includes the Genlock board, remote control and a disk with three sample animations and a picture, all for Genlock testing.

Operation couldn't be simpler. The Genlock is software-independent, meaning it will work with almost anything. In layman's terms, the Genlock functions by grabbing the ST's screen and determining which, if any, pixels are a color which has been set to absolute black (000 on the palette). If it is any color but that black, it is plotted on top of the video signal coming in from the composite video-in port. If the pixel is color 000, it does not plot that pixel, allowing the video image to show through the resulting "hole." Thus, to put Megabit Mouse on the screen, all I have to do is put him on a solid black background and make certain no part of him is color 000. If I need black on a character I'll use 001, which is close but doesn't vanish like 000.

Controlling the Genlock is simple. Its remote consists of three two-position

(to page 47)





12 Issues $28

$19 OFF THE COVER PRICE

12 Issues with Disk $79

NEW LOWER PRICE

BREAK AWAY FROM THE PACK


The world of ATARI-ST continues to grow by leaps and bounds, and ST-LOG is there every step of the way! We stand apart from the competition by offering more color, comprehensive reviews and in-depth features. SUBSCRIBE NOW!

☐ **12 Issues $28** MCSWW

☐ **12 Issues with Disk $79** DCSWW

ST LOG

☐ PAYMENT ENCLOSED ☐ BILL ME

CHARGE MY: ☐ VISA ☐ MASTERCARD

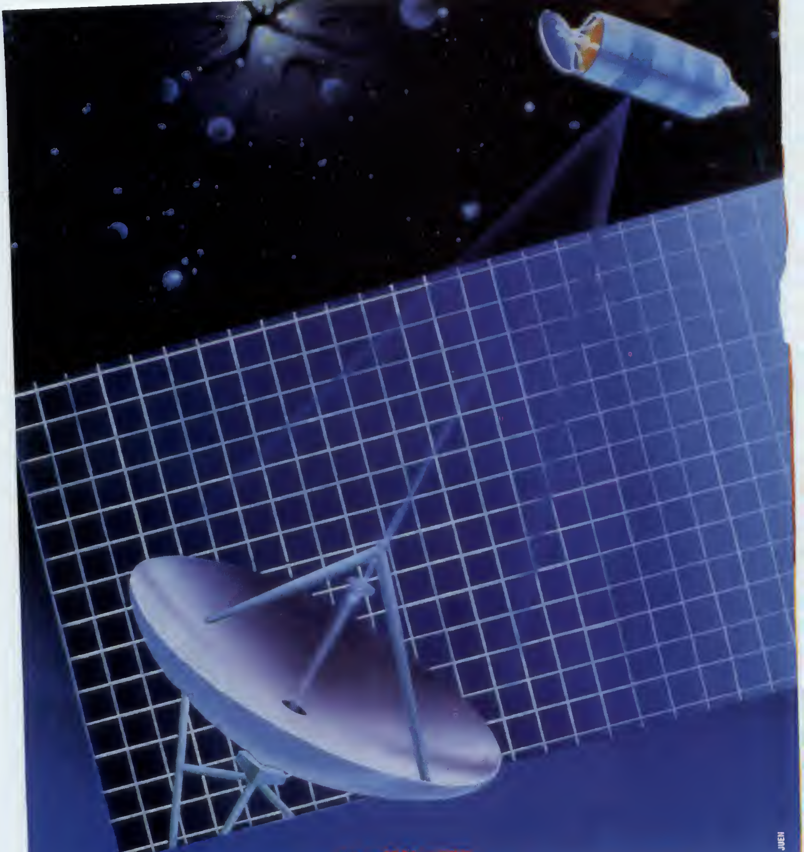
CARD # EXP SIGNATURE

NAME

ADDRESS

CITY STATE ZIP

MAKE CHECK PAYABLE TO L.F.P., INC., P.O. Box 16928, N. Hollywood, CA 91615. Offer expires Sept. 27, 1989


Make the DELPHI connection
As a reader of ST-LOG, you are entitled to take advantage of a special DELPHI membership offer. For only $19.95, plus shipping and handling ([illegible] off the standard membership price!), you will receive a lifetime subscription to DELPHI, a copy of the 500-page DELPHI: The Official Guide by Michael A. Banks, and a credit equal to one free evening hour at standard connect rates. Almost anyone worldwide can access DELPHI (using Tymnet, Telenet or other networking services) via a local telephone call.
To Join DELPHI
1. Dial 617-576-0862 with any terminal or PC and modem (at 2400 bps, dial 576-2981).
2. At the Username prompt, type JOINDELPHI.
3. At the Password prompt, enter STLOG
For more information, call DELPHI Member Services at 1-800-544-4005, or at 617-491-3393 from within Massachusetts or from outside the U.S.
DELPHI is a service of General Videotex Corporation of Cambridge, Massachusetts.
PHOTOGRAPHY: KEE HWA JUEN

Database DELPHI

BY ANDY EDDY

Last month we left off our coverage halfway through the *NEWS-WEATHER-SPORTS* menu, a section of DELPHI that brings you up-to-the-minute information. Let's finish up our coverage of this powerful area—then we'll get to some exciting developments taking place in the databases.

And now the news. . .

Again, here's the primary NEWS-WEATHER-SPORTS menu:

```
NEWS-WEATHER-SPORTS Menu:

Newsbrief                    Press Releases: Business Wire
Accu-Weather Forecasts       Sports
AP News Service              Quiz - Your News IQ
Astro Predictions            Today in History
CompuBug                     Views on News
Financial News               HELP
Kyodo News from Japan        EXIT
Movie News & Reviews
```

Astrology fans will enjoy the diversity provided by the *Astro Predictions* submenu. You can get "readings" covering a variety of situations: that day's horoscope, a week-long summary, compatibility between zodiac signs, even a guide for gift-giving. For an additional $12.50, you can get a hardcopy chart done up by Phyllis (DELPHI ID: PHYL) of your own personal astrological aspects.

Compubug is a regular column written about computers in the news, containing both serious subjects and satirical looks at our favorite hobby. It always provides provocative reading.

For serious investors or novice businessmen, the *Financial News* selection will help you brush up on your dollars and cents. Here you can find out the prices of gold and the current value of the dollar, get a listing of NYSE or AMEX stocks, get a listing of the most active stocks that day and much more. This is what the *Financial News* menu looks like:

```
Copyright (C) 1989 by the Associated Press
All rights reserved.

AP VIDEOTEX BUSINESS Menu:

General Business News     S&P Averages
Govt Economic Figs        NASDAQ
Dollar/Gold               Markets at a Glance
Board of Trade            Most Active Stocks
Money Supply              NYSE Lists
Stock Market Reports      AMEX List
Dow Jones                 EXIT
AP 60 Stocks
```

Because the Japanese have a great deal of influence on how American business runs these days, *Kyodo News from Japan* is important in keeping up with current affairs. This menu is broken down into a bunch of categories, such as bonds, commodities, government, money and stocks.

To prove this section isn't just devoted to serious topics, the next selection, *Movie News & Reviews*, lets you check out what's going on in films. Not only can you get reviews of what is currently in the theaters and peek in on what's in the works for future films, but you can also search through a database of reviews on previously released films. Especially with the popularity of video cassettes, this is a good way to pick and choose what movies are best for you.

Press Releases: Business Wire is a briefing room for up-to-date business dealings. There you can pick from various articles to keep you abreast of the happenings with the world's movers and shakers. Here's a sample of the *PR* menu:

```
Business Wire--Press Releases and News as It happens.
Copyright (c) 1987 Business Wire Inc.

 1  4-MAR  TIME INC. SSO WSAAES COMMUSICSTIOSS IAC. FOAM TIME WASSES IAC.
 2  4-MAS  FIAESTOME BSEAKS GAOUSO FOA AEW EXPAASIOS PSOJECT IS WILSOS, 1.C.
 3  3-MAR  AOUISOAY/EIGHT CSLIFOSSIAAS TO SE SONOAES FOS OUTATAADISG
           COAIAIAUIIOSS IS THE FIELS OF SLCOHOL-AELATES SEAVICES
 4  3-MGS  SIATENDO IMIENT ON WIMSISG LEGSL WAA AGAISST MSGAAUOX
 5  3-SAS  SOUISORY/OAEGOA SESLTS CSAE SSSOCIAIIOS TO SOLS PSESS COSFESEACS
           MAACH 7 SUPPORTIAG AEAATE SILL 903 SAA SSG77
 6  3-MAA  CARVUS AYAIEMS APPOIATS SEW PSESIOEST SSS CHIEF SSECUTIVE OFFICES
 7  3-MAA  AOC SEPOAIS TAISS QUAATEA FIMANCIAL SESULTS
 8  3-MAS  MAAITIME SESOAIS IATERAATIOAAL, SOUTSEAA SHIPBUILDIAG, SISCUSSIAG
           COAIAACT TO SUILS CAUISE SHIP
 9  1-MAR  SOAEAT M. SUTCNKO SSSUMES PSESISENCY SF AENAISSAACE GAS
10  3-MAA  TSACE PUACSSSES MAJOAITY IATEAEST IS VEATUNO COSP., S
           MAAUFSCIUSES OF SISKETIS SUPLICSTISS PSODUCTS/SYSTEMS

Enter Item Number, MORE, or EXIT:
```

The *Sports* selection duplicates the sporting scores and summaries that we talked about last month as part of the *AP News Service*. Again, you can get reports on any sport from here, as the section is broken down into many submenus.

When you choose the *Quiz Your News IQ* selection, it runs some questions past you to test your current-events knowledge. No, there's no prizes—unlike the FlipIt and Trivia Quest games that can be found on DELPHI—but no matter how well you do, it's fun *and* educational.

Today in History is a glance back at events that have taken place on this date. It's an interesting compendium of data from past historical happenings.

We close up the *NEWS-WEATHER-SPORTS* menu with a SIG—yes, this is a strange place for a SIG, but in fact the *Views on News* SIG has no better place than right here. Here's the menu you'll see when you enter:

```
VIEWS ON NEWS Menu:

Archives              Set Preferences
Conference            Who's Here
Forum (Messages)      Workspace
Features              Help (Hints)
MAIL (Electronic)     Exit
Poll

VIEWS ON NEWS>
```

Though we briefly discussed this area in the May 1988 *Database DELPHI*, just after this SIG was introduced, they've really polished it up. If you want to chat about current events or read through some cutting editorials, *Views on News* is the place to do it. Among the features they run is a column by Bob Fried called "Articles of Lasting Strangeness." This commentary on life always bring your emotions to the surface, generally resulting in a smile. Most people don't expect to get a chuckle out of a computer activity, but Fried usually breaks that misconception to pieces.

Free uploads

As we noted at the end of last month's column, DELPHI was in the process of reprogramming the database system to permit free uploading of files. If you have ever uploaded a file before, you know that DELPHI previously had you request free time from the SIG manager before uploading. In the meantime, both CompuServe and GEnie had adopted a policy of free uploads to make it easier on their users. Now DELPHI has followed suit, and the results are great.

Instead of uploading the file(s) to your workspace and then submitting it/them to the database (as was the previous procedure), you can now type "SUB" at your workspace or any database section, then follow the various prompts to not only submit the file, but also the list of keywords, the description and download names. Some additional bonuses have been implemented such as batch uploads for file groups, and the ability to edit the contents until the submission is complete.

Perhaps the most attractive feature is that you can do your upload process in pieces, and the system will keep the entry "in holding" until you are ready to submit it for inclusion in the databases. If you have the files uploaded but aren't sure of what to put in the description, you can hold off on that part until you are ready (or vice versa).

When you type "SUB" you will be prompted as to whether you want to use the new method or the old method, the latter of which requires that you have the files already in place in your workspace. If you pick the new method, you'll see an indicator that your billing has been shut off while you go about your business.

As we've noted, the most popular feature of any SIG is the ability to download files. If you have any files that you've acquired or written that are in the public domain, please take the time to upload them to the ST SIG. Your fellow DELPHI users will be happier for it!

Well, I see that our time is up for this month. Next issue, we'll tell you how you can use DELPHI to make travel plans.

Till next month, C U online.

THE YOU NEED DISK!

ONLY $9.95 EACH!

If you want to get the most out of ST-LOG, you're also going to want to get your copy of the disk. Each issue's disk contains all the exciting programs for that issue, including the programs whose listings could not be included due to space considerations. The ST-LOG disk version is truly an excellent software value. Order yours today!

MicroCheck ST, Game Cupboard, Animated GFA Input and more!

Line Attack, Outline Plus, Monochrome Gray II and more!

Picture-Puzzle, Sounds-A-Like, ChemCalc and more!

Ballbuster, MIDI Capture Safe Keeping and more!

Multi-Paint, Font ID Editor, Monochrome Gray and more!

Flag Trivia, Super Spool, Desk Switch and more!

YES I DO WANT THE DISK! ONLY $9.95 EACH

- ☐ ST-LOG February 1989 DISK
- ☐ ST-LOG March 1989 DISK
- ☐ ST-LOG April 1989 DISK
- ☐ ST-LOG May 1989 DISK
- ☐ ST-LOG June 1989 DISK
- ☐ ST-LOG July 1989 DISK

Name_____

Address_____

City_____ State_____ Zip_____

☐ Payment Enclosed—Charge My ☐ VISA ☐ MC

#_____ Exp._____

Signature_____

Add $1.50 postage and handling for each disk ordered.
Make check payable to: LFP, Inc. P.O. Box 67068, Los Angeles, CA 90067
California residents add 6.5%

PD Parade

BY GEORGE L. SMYTH

The program described here is available on this month's ST-LOG disk, in the databases of the ST-LOG ST users' group on DELPHI, and on BBSs and other online services throughout the country. Because this program is "shareware," anyone who enjoys it should send the requested donation to the author.

When I bought my Atari 520ST, I also purchased two pieces of software to get me started: a Pascal package and a graphics program. I bought the former because I needed to write some programs to replace those I had used with my old computer. I bought the latter because I was knocked out by the graphics capabilities of the ST. Two and a half years later, I am still intrigued with the ST's superior graphics power.

These graphics capabilities apparently also impressed David A. Pollette to the extent that he sat down and created this month's featured shareware program, *Guess-A-Sketch*, which combines the elements of a graphics program and a word game.

Guess-A-Sketch is designed to be played by two teams of two players each. The object of the game is to draw a picture representing a given word in such a way as to get a teammate to guess the word within the allotted 60 seconds. This is a difficult task, one that requires imagination and a quick drawing hand.

The opening screen requires each player to input his/her initials so that the computer can address each participant individually. A choice of word files is then presented. The game includes one word file; others may be created by the *Guess-A-Sketch* word file editor, which is available to those who send in their $10 registration fee.

Three different-sized game boards are available, one with 39 spaces for first-time players, one with 63 spaces for average players and one with 71 spaces for dedicated players.

The first time the game is played, it may be a good idea to use the practice mode, which allows game play without scoring. This mode is especially helpful if more than one player is new to the game.

The program then rolls the dice to determine which team will begin. One player from this team is identified as the artist and his teammate as the guesser. The person designated to guess then turns his/her head while the other clicks the mouse button. A word from each of the five topics is then displayed, one of which is chosen by the computer as the target word. The topics are labeled "person, place or animal," "object," "action," "difficult" and "miscellaneous." After ringing the bell and flashing the word eight times, the screen changes to a drawing board. At this time the guesser can face the screen and watch his teammate attempt to draw a picture suggestive of the word.

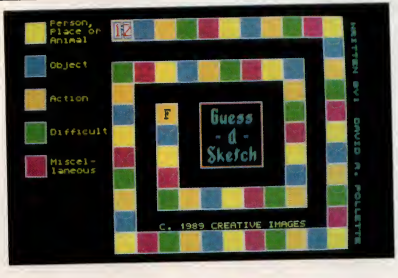
The individual drawing the picture has several tools to aid him. Besides having a pen with which to draw, the brush size can be changed, a straight line option is available, and the color of the pen may be altered. A fill option is also offered to speed the drawing process.

The lower left corner contains the outlined drawings of an ear, a pair of scissors and a plus sign. Clicking the right mouse button while pointing to one of these three areas brings up a large icon that indicates respectively, "sounds like," "cut" and "add." This is the level of play where imagination becomes important. A representation of the word can be drawn, portions of the word can be drawn, or its meaning can be altered and modified, as in the game Charade.

The upper-right corner of the screen contains the player's nemesis, the timekeeper. Starting at 60, it counts down, second by second, to zero. If the word is guessed before the minute has been counted, a key must be pressed to indicate success.

If the word is correctly identified, the computer rolls the dice to indicate how many spaces the team's token will be advanced. The successful team retains control of the mouse with the team members swapping draw/guess duties. If the team is not able to guess the word, control of the mouse is turned over to the opposing team. The winner is the first team to land exactly on the final space. This means that a team that is far behind still has a chance to catch up if their opponents run into bad luck with the dice.

An interesting feature of the game is the option to redraw the graphic that was created for the last word. Another nice feature is the status display. When "(S) =



Three different-sized game boards are available, one with 39 spaces for first-time players, one with 63 spaces for average players and one with 71 spaces for dedicated players.

STATUS" is visible at the bottom of the screen, choosing this option displays the number of the team that's winning, how many spaces ahead they are and the number of times each person has had control of the mouse.

Documentation that includes a full explanation of game play as well as registration information is included. The text also includes suggestions regarding etiquette of play, which one may or may not decide to follow.

I hope that this program proves to be as much fun for your group as it was for ours. And please support the hard-working shareware programmers by sending them your contribution..

George L. Smyth has a degree in psychology from West Virginia University and is currently employed as a programmer. He is the author of a series of tutorials on programming in FORTH.

ASSEMBLY

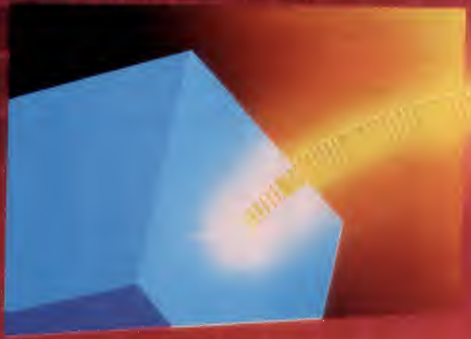
FILE HANDLING

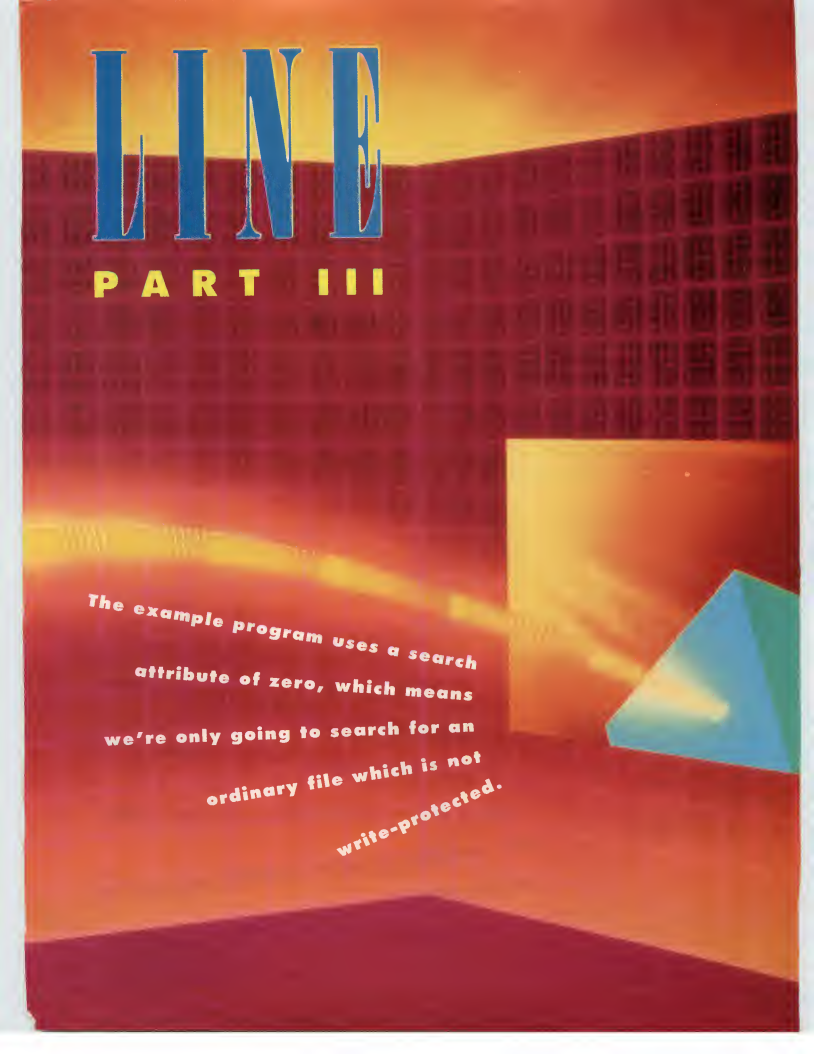
BY CHARLES F JOHNSON

In the last installment, we cut off the discussion of the example program just at the point where we had constructed a source and destination filename for our copy operation. (The source code for the example program was printed in the May '89 issue.) This month we'll continue where we left off, at the lines of code which read:

```
moveq   #0,d1           ; Search for normal files
lea     source,a5       ; Address of source filename
bsr     fsfirst         ; Search for it
bra.s   get.free
```

Remember that the source filename is stored at a location in the program labeled as *source*, and the destination filename is stored at *dest*.




LINE
PART III
The example program uses a search attribute of zero, which means we're only going to search for an ordinary file which is not write-protected.

Searching for a file

Now we're going to make sure that the file chosen by the user of our program actually exists, and also get some other important information about it by using the GEMDOS *Fsfirst* call (function number $4E). *Fsfirst* is a useful call; it returns just about every piece of pertinent data about a particular file, including its creation time and date, its size and its name.

But you say, "Wait, we already know the name of this file!" In the case of our current example program, this is true; but one of the real strengths of the *Fsfirst* call is that it can use the wildcard characters "*" and "?" to search for all the files in a given directory that match the specification. In a future column we'll be using the *Fsfirst* call (and its companion, *Fsnext*) to read a disk directory, but that, as I said, is for the future.

Getting back to our example program, we've set up a subroutine called *fsfirst*, which expects to be passed two parameters: the address of the file specification for which to search (which can contain the wildcard characters "*" and "?") and the search attribute. (Notice that the first "f" is *not* capitalized in the name of our subroutine, as it is in the name of the GEMDOS call.)

Before we use *Fsfirst*, we'll need to tell GEMDOS where to store all the information that will be returned from the call. The way to do this is with another GEMDOS call, *Fsetdta* (function number $1A)—and this is the first thing our *fsfirst* subroutine does. *Fsetdta* must be passed two parameters: the address of a 44-byte buffer (called the DTA, or disk transfer address buffer), which will be used to store the information returned from the *Fsfirst* call, and the function number $1A. In the example program, the DTA buffer is given the label *dta*.

When we call our *fsfirst* subroutine, the address of the file search specification is passed in register *a5*, and the search attribute is passed in *d5*. These registers will have remained intact throughout the *Fsetdta* call, since trap #1 preserves *a3-a6* and *d3-d7*. The search attribute word specifies the parameters shown in Figure 1.

To specify different search attributes, you simply set the bits for the attributes you wish to use. An alternative method is to add together the amounts in the "Value" column for the attributes you wish to use. The example program uses a search attribute of zero, which means we're only going to search for an ordinary file which is not write-protected.

After the *Fsfirst* call, *d0* will be set to zero if a file matching the search specification was found. If the search failed, or some other error occurred, *d0* will contain a negative number. So, after performing the GEMDOS *Fsfirst* call, our subroutine uses the *tst.l* instruction to set the N (negative) and Z (zero) flags in the condition code register according to the contents of *d0*, then returns with the usual *rts*.

Note that *rts* does not affect the condition codes, so when we return from the *fsfirst* subroutine we can simply *beq* (branch if equal to zero) to the code that follows. If the Z flag was not set, we print an error message and branch to our exit routine, labeled *byebye*.

If the Z flag was set upon returning from *Fsfirst*, we found a file that matches the search specification, and the DTA buffer contains all of this file's vital statistics, as shown in Figure 2.

Is there enough memory?

The next thing we need to do is find out whether the machine our program is running on has enough free RAM to read the entire source file into memory all at once. Our file copying program is designed to do its reading and writing in one pass; if there isn't enough memory to read in the whole source file at once, the example program just refuses to go any further. Come to think of it, there's a good project for the more adventurous and self-motivated among you—modifying the example program to copy a file of any size. The program would have to read and write the file in several passes. Any takers?

To find out whether there's enough memory for the copy operation to take place, we'll use yet another GEMDOS call: the much-feared and dreaded (and rightly so, as we'll explain in a moment) *Malloc* call.

Malloc (function number $48) is one of the most important calls in the GEMDOS library. It provides a way for ST applications to allocate memory that is protected from use by other applications. (Its companion/opposite call, *Mfree*, is discussed below.) *Malloc* expects to be passed only one parameter, a longword containing the number of bytes you wish to allocate. When you return from *Malloc*, *d0* contains the starting address of the block of reserved memory, or zero if the amount of memory requested exceeds the amount available.

However, if you pass a parameter of –1 (instead of a reserve amount) to *Malloc*, the call returns the size of the largest free block of memory in *d0*. The example program uses this version of *Malloc*, then compares the result with the size of the source file, which is contained in *dta*+30. If the size of free memory is smaller than the size of the source file, the *blo* (branch if lower than) instruction takes us to *no_memory*, which prints a message telling the poor user that he doesn't have enough memory to copy the file, and exits.

If there's enough free memory to load the entire source file, we use the *Malloc* call again. Since the size of the source file is contained in *dta*+26 (after the *Fsfirst* call, remember?), we can use the following code to reserve the memory we need:

```
move.l    dta+26,-(sp)
move      #$48,-(sp)
trap      #1
addq      #6,sp
```

If *d0* is not zero after this call, we branch over the code labeled *no_memory*

It is important to use the "long" form of the *tst* instruction when testing results from a file read or write call, because the number of bytes read or written can easily exceed a word value.

and continue on with our program. It's highly unlikely that this *Malloc* will fail—after all, we just used *Malloc(−1)* to determine that enough free memory existed—but better safe than sorry (an obnoxious truism that often holds very true in computer programming).

The trouble with *Malloc*

Now it's time for a short digression from our example program to discuss the problems with the GEMDOS *Malloc* call, mentioned briefly above. In the original ROM TOS (Version 1.0) and the newer TOS that was shipped with the Mega ST (Version 1.2), the GEMDOS *Malloc* call suffers from a particularly nasty bug. If an application calls *Malloc* more than a certain number of times (without using a corresponding number of *Mfree* calls), the ST gets confused.

If your application exceeds the critical number (which is usually somewhere around 40), strange things will start to happen. You may get spurious "out of memory" errors, files may refuse to load, and eventually you'll crash or lock up. The interval before the actual crash, however, is very dangerous indeed. If you try to write data to a disk when the system is in this confused state, that disk will almost certainly be corrupted beyond repair. The only solution when these symptoms start to appear is to immediately reboot your computer; once things get messed up in this way, they *stay* messed up.

The reason for this disastrous bug? GEMDOS maintains a list of all the blocks of memory allocated with *Malloc*, and when an application uses *Malloc* to reserve memory, another entry is simply added to the end of the current list. Unfortunately, the buffer that holds the list of *Mallocs* is of a fixed size, and GEMDOS does not check to see if it's already at the end when it adds a new entry. When the critical number is exceeded, new entries will actually write over other important GEMDOS data structures, wreaking havoc with the ST's file- and memory-management systems. (By the way, the *Malloc* bug is caused by the same problem which is responsible for the ST's well-known "40 folder" bug.)

Advance word has it that the new TOS 1.4, which will soon be released by Atari, fixes the *Malloc* bug and the 40-folder bug, by allowing true dynamic sizing of the memory allocation list. In the meantime, I recommend that you try to get a copy of Atari's FOLDRXXX.PRG, a program which runs from the AUTO folder and alleviates the problem by expanding the fixed memory list buffer to any size you specify. This program is available from Atari, or on many of the popular information services such as DELPHI, GEnie and CompuServe. If you're using a hard drive, this program is a necessity.

Luckily, since our example program uses only one *Malloc* call, it will not run into the *Malloc* bug. But if you ever write a program that needs to allocate memory in several chunks, you'll need to be aware of these potential problems.

Back in the saddle again

Okay, back to the example program. After successfully allocating a block of memory to read in the source file, we store the starting address of this memory in the variable *copy_buffer*, with the instruction:

```
move.l    d0,copy_buffer
```

We then print a message to let the user know his computer is going to be busy for a little while, and attempt to open the source file in "read only" mode. (The GEMDOS *Fopen* call was discussed in the January '89 *Assembly Line*.) We've written a subroutine called *open_file* that does this; the subroutine is passed the mode in *d0* and the address of the null-terminated filename in *a0*. Before returning, it uses the *tst.l* instruction to set the condition codes based on the results of the *Fopen* call. If the N flag is set, we use the *bmi* instruction to branch to the label *bad_open*, which prints an error message and branches to the exit code at *outta_here*. Otherwise, we save the file handle with the instruction:

```
move      d0,handle
```

Now (at last!) we're ready to start copying the selected file. The first thing we'll do is read in the entire source file, storing it in our allocated block of memory. To do this, we'll use the GEMDOS *Fread* call (function number $3F). Like the *Fopen* call, *Fread* was discussed in the January '89 *Assembly Line*. Our subroutine, called *read_file*, expects to be passed two parameters: the number of bytes to read in *d0*, and the address of the buffer into which to read it in *a0*. The example program calls the *read_file* subroutine with the following code:

```
move.l    dta+26,d0
move.l    copy_buffer,a0
bsr       read_file
```

The size of the source file is still contained in the DTA buffer, as returned from the *Fsfirst* call. So we just move the contents of *dta+26* to *d0* and the contents of *copy_buffer* (which holds the longword address of our allocated memory) to *a0*. Upon returning from the *read_file* subroutine, *d0* contains either a negative number (an error message) or the number of bytes successfully read from the file. Our program moves this value to *d7* temporarily, while it closes the file. This is necessary because the *Fclose* call alters *d0*. Then, after closing the file, we test *d7* to see if an error occurred during *Fread*. If *d7* contains a negative number we branch to the label *bad_read*, which prints an error message and branches to the exit code.

The moment you've been waiting for (almost)

At long last, we're coming to the payoff—the point where we can actually write the destination file and complete our copy program. But first (you knew there had to be one more delay, didn't you?), we have to discuss yet another bug in yet another GEMDOS call.

Up until now, all the GEMDOS file-handling calls that *Assembly Line* has discussed were the ones that deal with manipulating files that already exist. To create a new file, we'll use the GEMDOS *Fcreate* call (function number $3C). Unfortunately, there's a small but pesky bug in *Fcreate*, which causes it to sometimes create duplicate files (files with the same name), if the filename you're trying to create already exists. *Fcreate* is supposed to first delete the existing file when this occurs, but for some reason this doesn't always happen.

Therefore, before creating a new file under GEMDOS, it's a good idea to first explicitly delete any existing file with the same name. To do this, we use the *Fdelete* call (function number $41). *Fdelete* is passed only two parameters: the null-terminated filename you wish to delete and the function number itself. In the example program, we pass the address of the destination filename, located at *dest*. It doesn't matter if the file we're trying to delete doesn't already exist, so we ignore any errors from the *Fdelete* call.

The act of creation

Now we can create our destination file. The GEMDOS *Fcreate* call takes three parameters. First is the attribute word, which has the same format as the attribute word specified for the *Fsfirst* call, described above. By specifying different attributes, you can create subdirectories and "hidden" files with the *Fcreate* call.

Our example program uses zero for the attribute, which means that the file we create will be a normal, read/write file.

The second parameter passed to *Fcreate* is the longword address of the null-terminated name for the newly-created file, and the third parameter is the function number itself. As with all of our file-handling calls, we've written a subroutine to handle *Fcreate*, called *create_file*. The attribute word is passed to *create_file* in *d0* and the address of the filename in *a0*.

Fcreate returns either a valid file handle in *d0* or a negative error number. If we return from *create_file* with the N flag set, we branch to the label *bad_create*, which, as usual, prints an error message and exits. Otherwise, we save the file handle in *handle* and proceed to write the destination file.

The usage of the GEMDOS *Fwrite* call (function number $40) is identical to the *Fread* call. The only difference is the function number. The subroutine *write_file* handles the *Fread* call in our example program; it is passed the number of bytes to write in *d0* and the address of the buffer from which to write in *a0*. The code in our example program is very similar to the code used to read a file:

```
move.l    dta+26,d0
move.l    copy_buffer,a0
bsr       write_file
```

Upon returning from *write_file*, *d0* will contain either the number of bytes written without error or a negative error code. Just as with the *read_file* call, we move the result temporarily to *d7* while we close the open file. Then we use *tst.l d7* to see if an error occurred during the writing of the data, and branch to the appropriate error-handling code if necessary.

It is important to use the "long" form of the *tst* instruction when testing the results from a file read or write call, because the number of bytes read or written can easily exceed a word value. If the amount is larger than 32,767, a *tst.w* instruction will see it, erroneously, as a negative number.

Our example program doesn't handle one possible error that could occur while writing to a disk: running out of space on the disk. If this happens, GEMDOS does *not* report any error to you. It's up to you to make sure that the number of bytes that were actually written is the same as the number you wanted to write. After checking for errors, you should compare the value returned from *Fwrite* with the number of bytes you tried to write. If they are not equal, chances are that you ran out of room on the destination disk. In which case you should use *Fdelete* to delete the resulting partial file and let the user know that there's no more room on the disk.

Give back the memory!

When we're all finished with the copy operation, we still need to tie up one loose end before we exit the program. We have to give back to the system the memory we allocated with *Malloc*. The way to do this is with *Malloc*'s companion call, *Mfree*. *Mfree* takes two parameters: the longword address of the allocated memory you wish to free, and the function number, $49. The memory address must be the same as the value returned from *Malloc*. You can't *Mfree* memory that wasn't first allocated.

After the example program calls *Mfree*, we print a message asking the user to hit a key. When he/she does, we exit the program by calling the GEMDOS *Pterm0* function.

Stuff to do

Our example file-copying program is not perfect, by any means. For one thing, it assumes that you aren't trying to copy a file to the same disk (or subdirectory). Therefore, it's useless on a single-drive system, unless you use a RAMdisk to hold temporary files. You might try modifying the example program to prompt for a disk swap after reading the destination file. (Hint: The needed subroutines to do everything you need to do are already in the program.) Another good idea might be to allow retries after disk errors. Or to allow the user to make multiple copies without rereading the source file. The file handling subroutines in the example program will be useful in future examples, so be sure to save a copy of the source code.

Next time, we'll see how to modify our file-copying program to search a directory for files that match a "wildcard" specification, and introduce the concept of GEM alert boxes. Till then, code away! ▪

***Charles F. Johnson**, by using some as yet undiscovered laws of nature, has managed to find the time to be both a professional musician and a professional programmer. In his musical career, he has played with such artists as Chicago, George Duke, Al Jarreau and Stanley Clarke. His programming accomplishments include* Mouse-Ka-Mania, Desk Manager, ARC Shell *and, along with his partner, John Eidswoog,* G+ Plus *and* MultiDesk. *He and John are the owners of CodeHead software.*

Figure 1: ATTRIBUTE WORD

Bit	Value	
0	$00	Return files which have normal read/write access.
1	$01	Return files which are write-protected.
2	$02	Return "hidden" files (not visible on the desktop).
3	$04	Return "system" files (not visible on the desktop).
4	$08	Return the volume name of a disk.
5	$10	Return subdirectories.
6	$20	File has been written to and closed (also known as the "archive" bit).

Figure 2: DTA BUFFER STRUCTURE

Offset	
0-20	Reserved for internal use by GEMDOS.
21	File attribute.
22-23	Time of file creation (in standard GEMDOS format).
24-25	Date of file creation (in standard GEMDOS format).
26-29	Size of file, in bytes (longword).
30-43	Filename (8-character name, 3-character extension).

(from page 34)

switches and a dial. The dial is used in conjunction with the third switch and allows you to either fade the computer image on top of the video image or select between the two: one or the other or a ghostly half-and-half image in between. Switch one turns the Genlock on and off. Switch two allows you to flip between color and monochrome monitors (or toggling it quickly to mono and back will reset the computer). Switch three turns "keying" on and off, which effects how the dial operates.

Genlocked output can be either composite video or to an SC1224. The composite output, which works even with the Genlock off, seems to be at least as good as that provided by Practical Solutions' VideoKey. It videotapes nicely and looks good on a TV. The Genlocked output to the ST's own RGB monitor is both better and worse than the composite output. It's better because the clarity and sharpness of the RGB output surpasses that of the composite output. It's worse because the overall image is darker, less vibrant, and, because of the combination of multiple video signals, there's a slight quiver to the screen.

There are some hardware limitations. First, the Genlock does not work in monochrome because the ST's high resolution runs at 70 Hz, and composite video is 60 Hz, meaning the timing is incompatible. Second, this Genlock does not feature "overscanning" which means making the computer image fill the entire screen. The video image fills the entire screen, but the ST image stops at the screen borders. If you make the background color black, the borders become black, and the video image gets "boxed in" as well, somewhat solving that problem.

The third and most serious limitation is that the current model of the Genlock can only be used on a Mega ST, because the board contains a fan to keep the board from overheating and to prevent heat buildup inside the Mega and sits so high it could not fit in the less roomy cases of 520s and 1040s. Also, the small box with the monitor and other connectors hangs out of the processor bus access panel on the back of the Megas, an opening which does not exist on the 520 or 1040. JRI is testing a 520 and 1040 Genlock board, but at this point cannot guarantee if and when it will be available.

Problems? Few, all things considered. I've had no trouble Genlocking with any TV signal. The only trouble I had with Genlocking on the output from a VCR was with a few tapes that I suspect have copy protection on them. My tape of *2001: A Space Odyssey* produced a noticeable horizontal band of distortion on the Genlocked picture. The JRI Genlock manual warns about this.

Another small hitch appears when I try Genlocking on top of the composite output of my other ST. The Genlocked screen didn't seem to synch up properly, and would be vertically out of position many times. Interestingly, just plugging and unplugging the interconnecting video cable causes the vertical position to change, and with a few tries I can usually get them in synch. I'd guess this kind of problem can be corrected with a simple signal-synching box as used by TV stations, video-editing facilities, and even high school audio-visual classes.

The color/mono selection switch on the Genlock's remote control is neat, because with it you could plug an RGB monitor into the Genlock port and a monochrome monitor into the standard ST monitor port and toggle between them using that switch. This would eliminate the need for a switchbox, so I at first stowed my Monitor Master in the closet. I later realized that while this works, when the RGB monitor is plugged into the Genlock output jack, the brightness of the output is lower that usual, and the intensity of the colors is not as bright. This is noticeable on my old 1985 SC1224 made by Panasonic for Atari, this particular run of monitors can produce a much brighter screen than all later models, and is considered by most ST enthusiasts to be the best RGB Atari ever sold, and would be even more evident on the dimmer screens of later SC1224s.

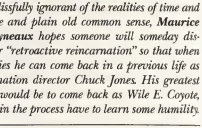
While this was fine when using the Genlock, I didn't like it for my other work, so I dragged out my Monitor Master again, hooked both monitors back to it, and plugged it into the Genlock's standard monitor out jack. I plug it into the other port only when I want to use the Genlock features. What we need is a switchbox which let's you select between two monitors *and* two different monitor ports! Interestingly, if you have a color/mono switchbox hooked up with a Genlock, toggling the switch on the switchbox itself will just cause the screen to go bananas, not reboot and switch resolutions. To switch monitors in this fashion you have to toggle the switches on both the switchbox *and* the Genlock!

The uses for a device like this are so numerous as to defy listing, covering applications from tinkering to professional output. You could make a cartoon character interact with real actors, superimpose titles over video, or even just get silly and doodle over your least favorite TV show.

If you are interested in the JRI Genlock, you can get more information by contacting them at JRI, P.O. Box 5277, Pittsburg, CA 94565; 415-458-9577. It's best to call before 1p.m. PST.

The sum of its parts

Any one of the products or techniques described above can be useful in jazzing up the graphics on your ST. Using two or more of them in combination can yield even more impressive results. The ST's graphics may not be able to compete head-to-head with those of a graphics workstation, but as you can see, the ST is not lacking in useful and powerful tools. Take those tools, add a little talent and imagination, and you may find that the combination will produce results you hadn't thought possible on an ST.■



*Blissfully ignorant of the realities of time and space and plain old common sense, **Maurice Molyneaux** hopes someone will someday discover "retroactive reincarnation" so that when he dies he can come back in a previous life as animation director Chuck Jones. His greatest fear would be to come back as Wile E. Coyote, and in the process have to learn some humility.*

THE COMPUKID CONNECTION:

Involving Young Children in ST Graphics Design

by D.A. BRUMLEVE

Note: *Due to the large size of the program associated with this article, it is available only on this month's disk or in the databases of the* ST-LOG *ST user's group on DELPHI.*

No matter where or in which culture they live, little children begin to scribble. The scribbles may seem formless and purposeless to adults, but the activity is purposeful to the child. In scribbling, there is a pattern to the placement of the scribbles and to the directions of the strokes. According to Rhoda Kellogg, a leading authority on children's art, there is a certain sequence in the development of drawing skills that is followed by all children. After experimenting with scribbling for some time, eventually the child begins to draw shapes, and, after that, outlines around the shapes. Soon after that comes the first attempt at what adults would consider representational art—drawings that depict a form that the adult perceives as an actual house or person or tree, etc. While children of the '80s still use crayons and paper to draw, they are also taking advantage of the powerful graphics capabilities of computers like the Atari ST. Does a small child use a computer drawing program the way he uses a crayon?

Not really, according to Michael Marks, director of Creative Discovery School in Champaign, Illinois. Marks has been using a color 520ST in an open classroom for several months. The computer is one of many activities in the classroom that the 19 preschool- and kindergarten-aged youngsters can choose. All of the students have had at least some experience with the computer; about one-third of them choose to use it on a regular basis. They use it without any adult help; all disks are prepared for auto-booting. Although they work independently, they rarely work alone. While one individual controls the mouse, a host of on-lookers participates in the computing experience, offering advice and commenting on progress. Among the most popular programs are not only games, but also a group of graphics design programs I have written: *KidGrid+* (MichTron), *Kidshapes* (freely-distributed) and a version of *Draw It!*, a program on this month's ST-LOG disk.

Drawing on the computer is "tapping a different cognitive process, having more to do with math, patterns and shapes," says Marks. "The computer is giving them an experience they could not get through drawing alone. The FILL option gives them a different appreciation and understanding of closed and open shapes." The children enjoy getting a reaction from the computer, seeing their own actions changing what is on the screen. "They love being able to quickly edit their work. Young children usually concentrate on outlines, and this gives them an opportunity to experiment, and it makes color and patterns much more important."

According to Marks, computer drawing develops different skills than drawing with a crayon. "If you can draw on a computer, it doesn't mean you can draw on paper, because mouse-drawing doesn't develop hand strength or proper grip, and the indirect eye-hand coordination is different." While this may seem to be negative, what it means is that a computer can be extremely helpful to children who have limited fine motor skills.

As Kellogg found with scribbling, Marks has observed a sequence to graphics work on the computer, but the sequence is different. First the children were interested in color; using *KidGrid+* and *Kidshapes*, they would change the colors of an existing picture. Then they began to work with color and shape patterns. In general, they are not yet putting these patterns together to create representations, yet all of these children are capable of representational art on paper. The recent introduction of *Draw It!* is moving these young users more toward representational computer artwork.

While very young children are more interested in the processes of filling and drawing than in the products they create, older children are more likely to use a computer as adults do. Children may draw for pleasure, but the similarity between a drawing program and a desktop publishing program is not overlooked. Children, like adults, may use a graphics design program to produce charts and graphs, posters, comic strips, newspapers, and the like. My oldest son, Danny (sixth grade), draws maps of story scapes to assist him in playing adventure games. Sons Willy and Joey (both age nine) have developed impressive portfolios of their computer art. Carl, a member of our children users' group, was inspired by his mother's needlepoint pillow and created, pixel by pixel, a picture of a rose. Seth, another child in our group, uses his drawings to illustrate his short stories. Children in a second grade classroom in our local public school are using their ST to produce illustrated stories and reports. An informal poll of 20 local children indicated that graphics design programs are second only to games in their importance as a child's enjoyment of a computer.

Most of these older children have learned to use sophisticated drawing programs that were designed for adults. In some cases, the child is able to use all the functions offered by the program. Other children have learned to use some of the functions, and they ignore the ones they haven't used. For some, the use of an adult-oriented application is appropriate, but for many, especially those under nine or ten, it is not. Many features of these programs, such as menus, file-selector boxes, technical terminology in alert boxes, brush and fill pattern editors, color palette editors, etc., may be a boon to adults, but they can add to a child's frustration and confusion.

A simple paint program

Draw It! is a children's paint program designed to provide a bridge for later work with *Neochrome*, *DEGAS*, *Art Director* and other more sophisticated graphics design software. The program provides experiences with computer art involving the two most basic graphics design operations: drawing and filling shapes with the mouse. Children as young as two may be able to enjoy working with *Draw It!*. The target audience for this program is between three and nine years of age.

Getting started

Separate high and low resolution versions of *Draw It!* can be found in archived (.ARC) files on your ST-LOG disk. First, de-ARC the files following the instructions found on the disk. Then if you have a monochrome monitor, copy the files MDRAW_IT.PRG and MSHOW_IT.PRG to a freshly formatted disk. If you have a color monitor, copy DRAW_IT.PRG and SHOW_IT.PRG in

Illustration - Fabienne Multz

stead. (The .LST files contain the GFA BASIC source code for the programs and are needed only if you want to examine the listing.) Although these files are small— about 40K total—when you save pictures created with the program, a large data file of about 115K will be created on your disk. The picture file for the color version is called DRAW_IT!.DAT; the monochrome version is MDRAWIT!.DAT. The programs will run properly from within a folder or from a hard disk, but any existing picture file must be in the same folder.

The monochrome and color versions have some differences. In the color version, you can draw and fill in any of 12 colors. In the monochrome version, of course, drawing is limited to black and white. In lieu of fill colors, the monochrome version offers a limited number of fill patterns, available only in black. In addition, the monochrome version offers a PRINT option. Most printers do not print colors well, so I have not included a PRINT option in the color version. If you happen to have a color printer, you can print the screen with a screen dump by pressing Alternate and Help at the same time. In each version, a white easel fills most of the screen, and the drawing options are above it.

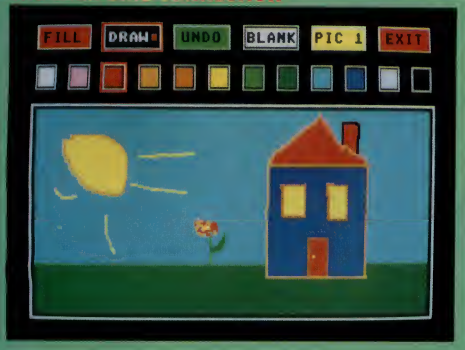
Using the program

Both versions offer two rows of options. The row at the top of the screen includes the following:

- DRAW sets the drawing design mode. If you move the mouse on the easel with either button down, a line will be drawn on the easel in the currently selected color. Three sizes of drawing "nibs" are available. Click the DRAW option repeatedly to see the nib sizes.
- BLANK completely erases a picture.
- UNDO erases the last drawing action. The UNDO function works even if you have selected colors or other options (even BLANK), but not if you have moved on to the next picture. Please note that if you click BLANK twice in succession, clicking UNDO at that point will restore the first blank easel, not your original picture.
- PIC # indicates the page number of the picture currently displayed. Click on PIC # to move to the next picture. Five pictures are available. If the PIC # option is clicked when picture No. 5 is on the screen, the first picture (PIC 1) is again shown.
- EXIT allows you to save your work and leave the program gracefully. Alert boxes are used to help prevent accidental exiting.

In the color version, the FILL option sets the filling design mode. If you click inside an enclosed shape on the easel with the FILL option selected, the currently selected color will fill the shape. The second row of options are the various colors with which you can draw or fill on the easel.

As mentioned, the monochrome version has a PRINT option. The color op-



tions (black and white) and a fill pattern box are positioned in a second row of options. Click once on the fill pattern box to select the filling design mode. Clicking repeatedly on the box will display all of the available fill patterns.

Addressing the needs of young computer artists

Draw It! is my seventh graphics design program for children. Some of the earlier programs have had a more successful user interface than others. I have had feedback from many users (and from parents and teachers of users), which has helped me develop programs that are easier to use. *Draw It!* is much easier to use than an adult paint program, and not only because it is limited to drawing and filling operations. Many additional features contribute to the kid friendliness of the program.

Undifferentiated mouse-button response is one of the most significant. This means that the program will react anytime the mouse is clicked on a target, no matter which button is used. It doesn't matter whether the child presses the left button or the right, or even both buttons; the result will be the same. Many children between three and nine cannot tell the difference between left and right, and this feature saves them much frustration.

Children tend to move the mouse about rather wildly, and accidental selections can occur. Usually, I avoid situations in which the child must hold the mouse button down while moving it ("dragging the mouse"). However, a primary purpose for *Draw It!* is to prepare the child for more sophisticated design applications, all of which require at least some dragging operations. I decided to include it, but special effort is taken in the programming to avoid inadvertent selections of options while drawing. If the mouse is dragged off the easel, and then moved back onto the easel, drawing continues from the point at which the easel is reentered.

Alert boxes confuse nonreaders, especially the first few times they run a program. Children who can read do not always take the time to do it. *Draw It!* uses alert boxes sparingly. Only the EXIT option has alert boxes that require a response. Even the BLANK option, which completely erases the screen, offers no warning to the user. To compensate, if a picture is erased unintentionally, the child can click UNDO and restore the picture (but only if no further drawing has been done and the page has not been changed). Error messages will occasionally alert the user to a problem. If the child clicks PRINT when no printer is connected to the computer, for example, a message will report the error accompanied by a declining scale. After a pause, the program will resume; no response is required of the user. While readers may find the message helpful, nonreaders will certainly realize that something isn't working as expected, and are likely to either ignore the problem or to consult an adult.

All mouse selections are indicated in two ways: A box surrounds the selected option and a sound is heard. The box indicator is typical of adult-level paint programs, so when the child moves on to more complicated programs, that feature will be familiar. The accompanying sound reinforces awareness that a selection has indeed been made.

In the color version, there are multiple indicators of the currently selected color and drawing mode. The current color option is surrounded by a box and the FILL and DRAW options both have indicators of that color. In addition, the mouse itself will be that color. In both the monochrome and color versions, the mouse serves as one indicator of the drawing mode. It takes the shape of a crayon for the drawing mode and of a paintbrush for the filling mode.

Young children simply cannot handle a file-selector box. *Draw It!* avoids its use entirely. If a picture data file exists in the same directory as the program, all five possible pictures are automatically loaded and shown on the title screen. If no picture file is found, the title sequence

DRAW IT! (monochrome) PRINTOUT

omits this display, and blank easels are presented on each of the five "pages" in the program. A further feature prevents problems during the loading sequence. Disks belonging to children are especially prone to damage. One scenario I have seen played out again and again is as follows: The child attempts to save his work, but removes the disk from the drive during the saving process. The filename remains on the disk, but it contains zero bytes. The program, when run again, attempts to load this zero-byte file. A major error occurs, and the program cannot continue.

To avoid this, *Draw It!* checks the size of the picture file before attempting to load it. If the size is not exactly what is expected, it deletes the file from the disk. The program then loads normally with blank easels, just as though there were no data file on the disk, and the child has an opportunity to save his work properly. It's worth noting that you shouldn't change the name of some other file on your *Draw It!* disk to MDRAWIT!.DAT or DRAW__IT!.DAT, but this would not be an easy task in any case: The Show Info option on your desktop will not allow you to put an exclamation mark in a filename; the picture file has been so named in order to prevent the easy substitution of other files.

Automatic loading has an additional value to a young child who may not quite understand that when a file is deleted on a disk, it cannot be retrieved. When all the pictures on the disk are loaded with the program, the child must consciously choose to erase a picture before beginning a new drawing. If he saves his new work, it will not be a total shock to him to find that the previous work is not loaded when he runs the program the next time. The artist is limited to five pictures because additional pictures would exceed a 520ST's memory capacity. My experience has shown, however, that five is a sufficient number of pictures for young children to work with. The process of drawing is much more important to them than the product, and, generally, they do not find it excruciatingly difficult to part with an old work when beginning a new one.

One other feature makes using the monochrome version of *Draw It!* less frustrating than it might have been. Most people who have worked with drawing programs have had the experience of filling a shape and then waiting an interminable time for the filling process to be completed so that work can resume. Certain fill patterns are especially likely to cause problems of this kind. These patterns have been avoided. Filling with white fill patterns is not allowed, and only a white area can be filled with the available black fill patterns.

The options of *Draw It!* are purposefully limited to reduce screen congestion. The options and easel share the same screen so that indicators of the current selections are continuously visible. While this prevents the easel from occupying a full screen, the advantages for the young child far outweigh the disadvantages, and there is still plenty of room on the easel for the development of a picture.

Involving your infant in ST graphics

The *Show It!* program, also available in monochrome and color versions, provides a slide show consisting of five pictures created with *Draw It!* While young children may enjoy displaying their own work with a slide show, *Show It!* is actually intended to involve their tiny siblings in computer graphics. *Show It!* is a slide show for newborn infants to two-year-old toddlers. Because children so young may have trouble clicking the mouse, keyboard input is used to advance the display to the next picture. If you have a baby in the house, he or she will love pressing (or kicking) the keys. A loud sound will bring the baby's attention to the monitor each time a key is pressed. To exit the program, simply click a mouse button. If you do not want your toddler or infant to exit the program, remove the mouse while the child works with the slide show.

My youngest son, Mickey, was born about the time I became reasonably proficient with *DEGAS*. I created for him an auto-booting slide-show disk consisting of drawings of his own familiar toys. Each toy was isolated on the screen against a white background, and the name of the toy was printed beside the picture ("Mr. Sailor Bear"). Babysitters, older siblings and doting relatives would run the slide show for Mickey and read each toy name aloud as Mickey kicked the keyboard to display yet another picture. Mickey especially liked pictures with faces (his bear, a doll, a brother, etc.). This was an enjoyable experience both for Mickey and his helper. One nice side effect was that the helpers learned to use the same names for his toys as I did, and could therefore communicate more effectively with him. Unlike the slide-show program I used, *Show It!* has the advantage of producing an auditory, as well as a visual, response each time a key is pressed. Now at two years of age, Mickey is thoroughly enjoying *Show It!*

Testing your printer for use with *Draw It!*

The monochrome version of *Draw It!* uses a command for a screen dump in order to print a picture. This procedure will provide an excellent copy of a picture from most dot-matrix printers. To test your printer's ability to accept a screen dump, first make sure your printer is on, connected and loaded with paper. Then, from your ST's desktop, press and hold the Alternate key and at the same time press the Help key. Your printer should kick on and print out a picture of the screen.

If the printout shows the entire screen, including the right edge, printouts can be made with MDRAW__IT.PRG without difficulty. If, however, the test printout omits the right edge of the screen, you will need to do the following:

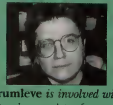
Copy the Control Panel accessory (CONTROL.ACC) that came with your ST onto your *Draw It!* disk. The *Draw It!* disk will become a boot disk for use whenever you work with the program. If your computer boots from a drive other than A, copy CONTROL.ACC to your boot disk.

Reboot your computer with the disk in Drive A. When the desktop appears, pull down the Desk menu and click on Install Printer. Change the Pixels/Line setting from the default of 1280 to 960. Now save your desktop.

Whenever you use *Draw It!*, reboot your computer from the boot disk. Your printer will automatically be set up to print a full-view screen dump. You will not need to change the Pixels/Line setting again, since that has been saved in a DESKTOP.INF file on your disk.

A problem with printouts involves the left-to-right adjustment of the picture. The picture should be centered on the printed page and the program is designed to make this happen. But if your printout is off center, you may want to adjust the paper. ■

—D. A. B.



D. A. Brumleve *is involved with children and computers in a variety of ways. The mother of five children ages 2 to 10, she serves as the adult facilitator of the Children's ST Users' Group in Urbana, Illinois. An avid programmer, she has developed a beginners' course in GFA BASIC and is the author of PreSchool Kid-Progs (MichTron) and numerous freely-distributed programs for young ST users.*

C-manship

BY CLAYTON WALNUM

It has always been my intention, after we covered as many GEM programming topics as necessary, to present in this column a complete application program. My plans were to cover the program over the course of several issues, with a new portion being presented in each installment, and at the end of the series, put the pieces together to form the complete application. I thought that this would be the best way to tie together everything we've been discussing for the last couple of years.

The problem with this idea has been finding the time to come up with a full-fledged GEM program. Such a project takes months of programming, and with my responsibilities as Executive Editor of *ANALOG* and STLOG, that time just hasn't been available to me.

I did, however, recently finish *MicroCheck ST*, which is the perfect program to use as a sample GEM application. So, this program is going to serve us double-duty. It is included in this issue as a regular feature, and we will also use it in the next few months of *C-manship*, to dig into the source code and see what makes it tick. Virtually every GEM topic we've discussed in the past is put to practical use in *MicroCheck ST*.

To understand these discussions, you will need a good background in GEM programming. I won't be reviewing topics extensively, but rather will be pointing out the ways in which the techniques we've studied are put to use in this program. For those of you who may have missed some of the *C-manships*, I will include, where appropriate, references to the past columns, so you'll know where to look for more information on a particular topic.

The listings

Listing 1 is the header file—created by the Resource Construction Set—for *MicroCheck ST*. Listing 2 is the first portion of the source code. In order for these listings to run properly, you need to have the file MICROCHK.RSC, which can be found on this month's disk version. For those of you who don't want to buy the disk version, I had originally planned to include in this month's column an ST BASIC program that would create this file for you. But due to space limitations, I was unable to include it at this time. I hope to supply that listing next month.

For those of you who do have this month's disk, the portions of *MicroCheck ST* included here in Listings 1 and 2 will compile and link with no problems, even though they aren't the complete program. Be forewarned, though, that when you run the program created from these listings, the only way to get out of it is to reboot your computer. The portion of *MicroCheck ST* that includes the Quit function is not presented this month.

Note: Some of the lines in Listing 2 end with a tilde (~). The tilde means that the line "wraps around" onto the next line of the listing. The two lines should be typed as one, leaving out the tilde.

The discussion

As I said above, Listing 1 is a file that was created by the Resource Construction Set. It contains nothing more than a series of *#defines* that equate the object and tree ID numbers of our resource with names that are easier to remember and that make for better reading code (see the April '87 *C-manship*). There's not much to say about this listing, except that you'll see every name there used somewhere in the *MicroCheck ST* source code (though not necessarily in the portion being presented this month).

Listing 2 is a small section of the *MicroCheck ST* source code. It is only about ⅛ of the full program, which gives you some idea of how large a full-GEM application may be. Although GEM is a great boon to the end user, whatever conveniences he gets are passed on to the programmer as extra work. A large portion of a GEM program deals with handling GEM rather than getting down to the business of the application itself. Setting up and handling dialog boxes, windows and menu bars takes many lines of code.

At the very top of the listing we have some *#includes*, which tell the compiler to add these files into our program at compilation time. We've discussed these files before. Note that the MICROCHK.H file is also included here.

Below the *#includes* we define some constants of our own. Just as we saw with the MICROCHK.H file, anytime we can replace a number, which tends to be cryptic, with a name, we'll be making our programming task easier and our resultant code more readable. Which makes the most sense to you: 0X1E01 or CNTL-A?

Below that we have the usual GEM global arrays. Every GEM application has to provide these storage areas.

Next we declare some of our variables. The array *msg__buf[]* will be used to store messages sent to us by GEM. The array *purs[]* is used in a conversion function not shown in this month's listing. There's also a long list of integers and character arrays. I won't spend a lot of time now explaining what each one is. We'll talk about them as they appear in the listings each month. If you look through the list of integers being declared, though, you'll see a lot of variable names you've run across before—variables that are needed to handle GEM's many functions.

Take a look at the pointers of type OBJECT declared below the character strings. If you've been following *C-manship* closely and keeping up on your studies, you'll know that these pointers will contain the addresses of each of the trees that make up our full resource tree (see the May '87 *C-manship*).

A little further down, you'll see a structure named *check*. This structure has storage areas for each piece of data we need for a checking account transaction: the check number, the payee, a memo field, the date the check was written, the amount of the check, and a field to indicate if the check has been cancelled (processed by the bank).

Of course, a checking program that'll hold enough data for only one check is useless. That's why our next step in setting up our data is to create an array of these structures—the arrays named *checks[]* and *srch__checks* []. The former will hold all the transactions for a particular month and the latter will hold all the transactions that match the search criterion when a search of the account is performed. The pointer **cur__chk-strc*, will be used to keep track of which of the two check structures we're currently using.

Finally, the last item declared before the program begins is the pointer **ob__tedinfo*, which is a pointer to a TEDINFO structure. Hopefully, you'll remember that a TEDINFO structure is used to hold the information we need for an editable text field in a dialog box (see the May '87 *C-manship*).

Function *main( )*

Every program is made up of three main sections: initialization, the program

itself and the job-end section (cleanup). The function *main()* breaks these three sections into six easy-to-follow steps. The functions *appl_init()* and *open_vwork()* initialize our GEM application (not the program, mind you, just GEM). The function *do_mcheck()* is the controlling function for *MicroCheck ST*—where *main()* handles the three sections of setting up the GEM operating system, *do_mcheck* does the same for our actual program. Finally, the functions *v_clsvwk()*, *Setcolor()* and *appl_exit()* perform the job-end duties, closing down our workstation and returning the GEM desktop to the same condition it was when we left it.

Function do_mcheck ( )

The function *do_mcheck()* begins by setting the ST's colors the way we want them and checking to see if the user has run the program in the proper resolution. If the resolution is okay, we set the mouse to an arrow and initialize some strings and variables. Then we get the system date with a call to *get_date()*.

The next step in our program initialization is to load the resource file and get the addresses of each of the trees that make up the resource. First, we check to see that the file MICROCHK.RSC is on the disk. (It must be in the same directory as the main program.) If it's not, we warn the user of the error and return to the desktop. If the resource file is available, we load it and get the addresses of each of the trees. Remember that each of these trees makes up one of the GEM forms—such as a dialog box or menu bar—that we will be using to get data to and from the user.

After we load the resource file, we bring up the menu bar with a call to *menu_bar()* and set the entries in the menus appropriately with a call to *set_menu_entries()*.

Now all we have to do is set up our windows, and we're all set (see the July/August '87 *C-manship*). In *MicroCheck ST*, there are actually two windows in use, although only one of them is visible. The invisible window has no parts (sliders, arrows, etc.) and covers the entire screen area. I use it to get redraw messages for portions of the screen that are not covered by the visible window.

After we set up the windows, we send program execution to the function *get_event()*, which routes the events our application receives from the user to the proper sections of the program.

Eventually, the user will indicate that he wishes to quit the program. When he does, the last portion of *do_mcheck()* removes the menu bar, closes and deletes the two windows, and returns the memory used by our resource tree back to the system. (Remember: this month's program segment doesn't let you quit.)

Function get_event( )

As you should already know, a GEM application program receives its instructions from the user by way of "events." There are many types of events, handling not only GEM constructions such as windows, menu bars and dialog boxes, but also the mouse and the keyboard. In *MicroCheck ST* we are interested in three main types of events: keyboard, mouse and GEM message events.

If you take a look at the function *get_event()* in Listing 2, you'll see that it takes only a small amount of source code to retrieve and route the events. Basically, all we have to do is get the event, figure out what type it is and pass it on.

To get the events, we use the unwieldy and complicated function *evnt_multi()* (see the June '87 *C-manship*). The integer *event* will hold the event number, which we'll test in three different *if* statements, each of which will route its event to the proper function.

The three functions, *handle_key()*, *handle_messages()* and *handle_button()*, process keyboard, message and mouse-button events, respectively. Notice that, at the end of Listing 1, these functions are represented by "stubs"; that is, functions that do nothing except provide a label for the linker. Without these stubs, you would not be able to link the program successfully. (The actual functions will be presented next month.)

Function set_menu_entries( )

The function *set_menu_entries()* in Listing 1 disables any entries in the menu bar that we don't want the user to have access to (see the June '87 *C-manship*). For example, until an account has been loaded, it's not possible to perform a search on the checks in the account. Rather than having to give the user an error message when he clicks on the Search option, we just make the option unavailable to him. The function *set_menu_entries()* is only one of three functions in *MicroCheck ST* that enable and disable menu options based on the program's current mode.

Functions calc_vslid( ) and calc_hslid( )

The functions *calc_vslid()* and *calc_hslid()* set the sizes and positions of the window's vertical and horizontal sliders (see the May '88 *C-manship*). This can be a confusing process, but one that is essential to the proper handling of windows. Assuming you understand how the sliders work, the only thing worth noting in these functions is found in *calc_hslid()*, where the flag *left* is used to determine the position of the horizontal slider. This method is used because this slider can be in only one of two positions—all the way to the left or all the way to the right.

Function open_vwork

Now we come to a function that *C manship* readers have seen dozens of times, *open_vwork()*. Anyone who doesn't know that this function sets up a virtual workstation, a necessity for a GEM application, should go back to square one and do some heavy reviewing.

Function get_date( )

Finally, the last function presented this month, *get_date()*, is responsible for retrieving and formatting the date from the ST's clock (see the September '88 *C-manship*). In this function, we're setting up two different strings, *date_buf[ ]* and *cur_date()*. The former will be in the format mm/dd/yy and will be displayed in a date button at the bottom of the screen. The latter will be in the format mmddyy and will be used as the default date for the check-entry dialog box.

Final notes

When you run this segment of *MicroCheck ST*, you'll find on the screen a half-working menu bar and a window with a blank work area. In addition, the information buttons that are normally displayed at the bottom of the screen (see the illustrations accompanying the *MicroCheck ST* article in this issue) will be missing. This is normal and has to do with the fact that this portion of the program is not set up yet to process GEM message events.

In closing, I would strongly urge those of you who wish to follow this in-depth look at a GEM application program to purchase this month's ST-LOG disk version. The complete *MicroCheck ST* can be found there, and I believe that it will help you better understand our discussions if you're familiar with the program. I will, however, try to make these columns as "freestanding" as possible, so that those who do not wish to purchase the disk will be able to follow along.

Next time, we'll look at another chunk of *MicroCheck ST*. ■

C-MANSHIP
Listing 1:C

```
#define NEWADIAL 0          /* TREE */
#define MENUBAR 1           /* TREE */
#define FILEDIAL 2          /* TREE */
#define DATEDIAL 3          /* TREE */
#define SRCHDIAL 4          /* TREE */
#define NEWOK 8             /* OBJECT in TREE #0 */
#define NEWCANCL 9          /* OBJECT in TREE #0 */
#define ABOUT 10            /* OBJECT in TREE #1 */
#define NEWACCNT 19         /* OBJECT in TREE #1 */
#define OPENMBR 20          /* OBJECT in TREE #1 */
#define CLOSEMBR 21         /* OBJECT in TREE #1 */
#define QUIT 23             /* OBJECT in TREE #1 */
#define ENTER 26            /* OBJECT in TREE #1 */
#define SEARCH 27           /* OBJECT in TREE #1 */
#define RECONCIL 28         /* OBJECT in TREE #1 */
#define PRNTWIND 32         /* OBJECT in TREE #1 */
#define PRNTREG 33          /* OBJECT in TREE #1 */
#define NEWYEAR 35          /* OBJECT in TREE #1 */
#define NEWDATE 36          /* OBJECT in TREE #1 */
#define DESK 3              /* OBJECT in TREE #1 */
#define FILEBAR 4           /* OBJECT in TREE #1 */
#define CHECKS 5            /* OBJECT in TREE #1 */
#define PRINT 6             /* OBJECT in TREE #1 */
#define UTILITY 7           /* OBJECT in TREE #1 */
#define FILENAME 3          /* OBJECT in TREE #2 */
#define FILEOK 1            /* OBJECT in TREE #2 */
#define FILECANC 2          /* OBJECT in TREE #2 */
#define NWDATE 2            /* OBJECT in TREE #3 */
#define DATEOK 3            /* OBJECT in TREE #3 */
#define DATECANC 4          /* OBJECT in TREE #3 */
#define SRCHOK 5            /* OBJECT in TREE #4 */
#define SRCHCNCL 4          /* OBJECT in TREE #4 */
#define NEWNAME 2           /* OBJECT in TREE #0 */
#define NEWADDR 3           /* OBJECT in TREE #0 */
#define NEWCITY 4           /* OBJECT in TREE #0 */
#define NEWSTATE 5          /* OBJECT in TREE #0 */
#define NEWZIP 6            /* OBJECT in TREE #0 */
#define NEWBALNC 7          /* OBJECT in TREE #0 */
#define MNTHFROM 7          /* OBJECT in TREE #4 */
#define MNTHTO 8            /* OBJECT in TREE #4 */
#define NUMFROM 9           /* OBJECT in TREE #4 */
#define NUMTO 10            /* OBJECT in TREE #4 */
#define AMNTFROM 11         /* OBJECT in TREE #4 */
#define AMNTTO 12           /* OBJECT in TREE #4 */
#define PAYEFROM 13         /* OBJECT in TREE #4 */
#define MEMOFROM 14         /* OBJECT in TREE #4 */
#define CHKCAN 29           /* OBJECT in TREE #1 */
#define CANCDIAL 5          /* TREE */
#define CANCOK 3            /* OBJECT in TREE #5 */
#define CANCCANC 4          /* OBJECT in TREE #5 */
#define JAN 6    /* OBJECT in TREE #5 */
#define FEB 7    /* OBJECT in TREE #5 */
#define MAR 8    /* OBJECT in TREE #5 */
#define APR 9    /* OBJECT in TREE #5 */
#define MAY 10          /* OBJECT in TREE #5 */
#define JUN 11          /* OBJECT in TREE #5 */
#define JUL 12          /* OBJECT in TREE #5 */
#define AUG 13          /* OBJECT in TREE #5 */
#define SEP 14          /* OBJECT in TREE #5 */
#define OCT 15          /* OBJECT in TREE #5 */
#define NOV 16          /* OBJECT in TREE #5 */
#define DEC 17          /* OBJECT in TREE #5 */
#define RECNDIAL 6      /* TREE */
#define ENDBAL 2        /* OBJECT in TREE #6 */
#define ENDBOK 3        /* OBJECT in TREE #6 */
#define ENDBCANC 4      /* OBJECT in TREE #6 */
#define CANCSTRG 1      /* OBJECT in TREE #5 */
#define MZERO 18        /* OBJECT in TREE #5 */
#define CHEKDIAL 7      /* TREE */
#define CHKNAME 1       /* OBJECT in TREE #7 */
#define CHKSTREE 2      /* OBJECT in TREE #7 */
#define CHKCITY 3       /* OBJECT in TREE #7 */
#define DATE 5          /* OBJECT in TREE #7 */
#define NUMBER 4        /* OBJECT in TREE #7 */
#define PAYEE 6         /* OBJECT in TREE #7 */
#define MEMO 8          /* OBJECT in TREE #7 */
#define CHKDONE 10      /* OBJECT in TREE #7 */
#define CHKNEXT 9       /* OBJECT in TREE #7 */
#define AMOUNT 7        /* OBJECT in TREE #7 */
#define CHKCANCL 11     /* OBJECT in TREE #7 */
#define RPRTDIAL 8      /* TREE */
#define RECENDB 3       /* OBJECT in TREE #8 */
#define RECSUBT 8       /* OBJECT in TREE #8 */
#define RECCHKS 5       /* OBJECT in TREE #8 */
#define RECDEPS 10      /* OBJECT in TREE #8 */
#define RECBALSH 13     /* OBJECT in TREE #8 */
#define RECOUTCH 4      /* OBJECT in TREE #8 */
#define RECOUTDP 9      /* OBJECT in TREE #8 */
#define RECBALIS 15     /* OBJECT in TREE #8 */
#define RECDIF 18       /* OBJECT in TREE #8 */
#define RECOK 19        /* OBJECT in TREE #8 */
#define LKMNDIAL 9      /* TREE */
#define CANMNTH 2       /* OBJECT in TREE #9 */
#define CHKAUTO 30      /* OBJECT in TREE #1 */
#define NEWMNTH 24      /* OBJECT in TREE #1 */
#define PAID 12         /* OBJECT in TREE #7 */
#define IMPORT 37       /* OBJECT in TREE #1 */
#define SRTODIAL 10     /* TREE */
#define DBCNT 3         /* OBJECT in TREE #10 */
#define DBTOT 5         /* OBJECT in TREE #10 */
#define CRCNT 7         /* OBJECT in TREE #10 */
#define CRTOT 9         /* OBJECT in TREE #10 */
#define SRTOOK 10       /* OBJECT in TREE #10 */
```

C-MANSHIP
Listing 2: C

```
/*************************************************************
*                    MICROCHECK ST                           *
*                  by Clayton Walnum                         *
*                                                            *
*              Copyright 1989 by ST-LOG                      *
*                Developed with Laser C                      *
*************************************************************/

#include <stdio.h>
#include <osbind.h>
#include <gemdefs.h>
#include <obdefs.h>
#include <fcntl.h>
#include "microchk.h"

#define WA_UPPAGE    0
#define WA_DNPAGE    1
#define WA_UPLINE    2
#define WA_DNLINE    3
#define WA_LFPAGE    4
#define WA_RTPAGE    5
#define BOLD         1
#define LIGHT        2
#define TRUE         1
#define FALSE        0
#define YES          1
#define NO           2
#define LEFT_BUTTON  0x0001
#define BUTTON_DOWN  0x0001
#define NUM_CLICKS   2
#define PARTS        NAME|INFO|UPARROW|DNARROW|VSLIDE|FULLER|CLOSER|HSLIDE
#define NUM_COLUMNS  93
#define MED          1
```

```c
#define MATCH          0
#define REC_LENGTH     117
#define FROM_BEG       0
#define FROM_CUR_POS   1
#define FAILED         (-1)
#define DFLT_DRV       0
#define VISIBLE        1
#define MEDIUM         1
#define HIGH           2
#define CHAR_AVAIL     -1
#define CONSOLE        2
#define ESCAPE         27
#define CNTL_A         0x1e01
#define CNTL_B         0x3002
#define CNTL_C         0x2e03
#define CNTL_D         0x2004
#define CNTL_E         0x1205
#define CNTL_G         0x2207
#define CNTL_I         0x1709
#define CNTL_M         0x320d
#define CNTL_N         0x310e
#define CNTL_O         0x180f
#define CNTL_P         0x1910
#define CNTL_Q         0x1011
#define CNTL_R         0x1312
#define CNTL_S         0x1f13
#define CNTL_W         0x1117
#define CNTL_Y         0x1519

int work_in[11], work_out[57], contrl[12],
    intin[128], ptsin[128], intout[128], ptsout[128];

int msg_buf[8];

long pwrs[] = { 1, 10, 100, 1000, 10000, 100000, 1000000, 10000000 };

int handle, dum, file, key,
    fullx, fully, fullw, fullh, wrkx, wrky, wrkw, wrkh,
    w_h1, w_h2, res, full, num_trans, charw, charh, curchknum,
    num_deps, num_chks, loaded, all_done, mouse_x, mouse_y,
    num_clicks, edit_top, left, start_mnth, end_mnth, mnth, srch_trans,
    start_num, end_num, cur_count, cur_top, search, saved, canceling,
    month, full_draw, oldcolr;

int zero = 0;

char filename[64], chkname[30], chkstreet[30], chkcity[50],
     date_but[10], bal_but[10], trans_but[4],
     check_but[4], dep_but[4], mnth_but[10], acct_name[64],
     monthfile[64], cur_chk_num[6], cur_date[7], future_use[40],
     cancmnth[5], chtot[20], dptot[20], chcnt[10], dpcnt[10];
char windname[64];
char noacct[] = "No account open-d";

char canc[] = "CANCEL CHECKS";
char newm[] = "  NEW MONTH  ";

char *months[] = { "Month 0", "January", "February", "March", "April",
                   "May", "June", "July", "August", "September", "October",
                   "November", "December" };

char spaces[] = "                                             ";
char infotext[] = "  Number   Amount    Payee                             Mem~
o                        Date";
char *string, *srch_payee, *srch_memo;
char rule[] = "-----------------------------------------------------------~
-----------------";

long balance, start_amnt, end_amnt;

OBJECT *menu_addr, *check_addr, *newacct_addr, *newfile_addr,
       *newdate_addr, *srchdial_addr, *cancdial_addr, *recndial_addr,
       *rprtdial_addr, *lkmndial_addr, *srtodial_addr;

FILE *acctfile, *mfile;

char *get_tedinfo_str ();
FILE *opn_nw_auto ();
long str_to_long ();

struct check {
   char number[5];
   char payee[31];
   char memo[31];
   char date[9];
   long amount;
   char cancel[2];
};

struct check checks[500];
struct check srch_checks[1000];
struct check *cur_chk_strc;

TEDINFO *ob_tedinfo;


main ()
{
   appl_init ();              /* Initialize application.    */
   open_vwork ();             /* Set up workstation.        */
   do_mcheck();               /* Go do MicroCheck.          */
   v_clsvwk (handle);         /* Close virtual workstation. */
   Setcolor ( 2, oldcolr ); /* Reset color register.      */
   appl_exit ();              /* Back to the desktop.       */
}


do_mcheck()
{
   oldcolr = Setcolor ( 2, -1 );
   Setcolor ( 2, 0x005 );
   if ( (res = Getrez ()) != HIGH && res != MEDIUM )
      form_alert(1,"[0][MicroCheck ST runs|only in high or medium |resoluti~
on.][OK]");
```

C-manship

PROGRAM LISTINGS

```
    else {
       graf_mouse ( ARROW, &dum );
       strcpy ( acct_name, "NONE" );
       strcpy ( cur_chk_num, "0000" );
       balance = 0;
       month = -1;
       edit_top = cur_top = num_trans = num_chks = num_deps = 0;
       left = saved = TRUE;
       search = canceling = full_draw = FALSE;
       cur_chk_strc = checks;
       get_date ();

       if ( !rsrc_load ( "MICROCHK.RSC" ) )
          form_alert ( 1, "[1][MICROCHK.RSC missing!][Okay]" );
       else {
          rsrc_gaddr ( R_TREE, MENUBAR, &menu_addr );
          rsrc_gaddr ( R_TREE, CHEKDIAL, &check_addr );
          rsrc_gaddr ( R_TREE, NEWADIAL, &newacct_addr );
          rsrc_gaddr ( R_TREE, FILEDIAL, &newfile_addr );
          rsrc_gaddr ( R_TREE, DATEDIAL, &newdate_addr );
          rsrc_gaddr ( R_TREE, SRCHDIAL, &srchdial_addr );
          rsrc_gaddr ( R_TREE, CANCDIAL, &cancdial_addr );
          rsrc_gaddr ( R_TREE, RECNDIAL, &srecndial_addr );
          rsrc_gaddr ( R_TREE, RPRTDIAL, &rprtdial_addr );
          rsrc_gaddr ( R_TREE, LKMNDIAL, &lkmndial_addr );
          rsrc_gaddr ( R_TREE, SRTDDIAL, &srtdodial_addr );
          menu_bar ( menu_addr, TRUE );
          set_menu_entries ();
          wind_get ( 0, WF_WORKXYWH, &fullx, &fully, &fullw, &fullh );
          w_h1 = wind_create ( 0, fullx, fully, fullw, fullh );
          w_h2 = wind_create ( PARTS, fullx, fully, fullw, fullh );
          wind_set ( w_h2, WF_NAME, noacct, 0, 0 );
          wind_set ( w_h2, WF_INFO, infotext, 0, 0 );
          wind_open ( w_h1, fullx, fully, fullw, fullh );
          wind_open ( w_h2, fullx, fully, fullw, 316 - 162*(res==MED) );
          calc_vslid ( 1 );
          calc_hslid ( NUM_COLUMNS );
          full = FALSE;
          loaded = FALSE;

          get_event ();

          menu_bar ( menu_addr, FALSE );
          wind_close ( w_h2 );
          wind_delete ( w_h2 );
          wind_close ( w_h1 );
          wind_delete ( w_h1 );
          rsrc_free ();
       }
    }
}


get_event ()
{
    int h, event;

    all_done = FALSE;

    while ( !all_done ) {
       event = evnt_multi ( MU_KEYBD|MU_MESAG|MU_BUTTON, NUM_CLICKS,
                            LEFT_BUTTON, BUTTON_DOWN,
                            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, msg_buf, 0, 0,
                            &mouse_x, &mouse_y, &dum, &dum, &key, &num_clicks );

       if ( event & MU_KEYBD )
          handle_keys ();

       if ( event & MU_MESAG )
          handle_messages ();

       if ( event & MU_BUTTON )
          handle_button ();
    }
}


set_menu_entries ()
{
    menu_ienable ( menu_addr, CLOSEMBR, loaded );
    menu_ienable ( menu_addr, OPENMBR, !loaded );
    menu_ienable ( menu_addr, MEMACCNT, !loaded );
    menu_ienable ( menu_addr, QUIT, TRUE );
    menu_ienable ( menu_addr, ENTER, loaded );
    menu_ienable ( menu_addr, SEARCH, loaded );
    menu_ienable ( menu_addr, CHKCAN, loaded );
    menu_ienable ( menu_addr, MEMMNTH, loaded );
    menu_ienable ( menu_addr, RECONCIL, loaded );
    menu_ienable ( menu_addr, PRNTWIND, loaded );
    menu_ienable ( menu_addr, PRNTREG, loaded );
    menu_ienable ( menu_addr, MEMYEAR, !loaded );
    menu_ienable ( menu_addr, CHKAUTO, loaded );
    menu_ienable ( menu_addr, NEWDATE, TRUE );
    menu_ienable ( menu_addr, IMPORT, !loaded );
}


calc_vslid ( line_cnt )
int line_cnt;
{
    int lines_avail, vslid_siz, pos;

    wind_get ( w_h2, WF_WORKXYWH, &wrkx, &wrky, &wrkw, &wrkh );
    lines_avail = wrkh / charh;
    vslid_siz = 1000 * lines_avail / line_cnt;
    wind_set ( w_h2, WF_VSLSIZE, vslid_siz, 0, 0, 0 );
    pos = (int) ( (float)(cur_top) /
          ( (float)(line_cnt - lines_avail) ) * 1000;
    wind_set ( w_h2, WF_VSLIDE, pos, 0, 0 );
}


calc_hslid ( col_cnt )
int col_cnt;
{
```

```
    int cols_avail, hslid_siz, pos, lft;

    if ( lft )
       lft = 0;
    else
       lft = 16;
    wind_get ( w_h2, WF_WORKXYWH, &wrkx, &wrky, &wrkw, &wrkh );
    cols_avail = wrkw / charw;
    hslid_siz = (int) ((1000L * (long) cols_avail) / (long) col_cnt);
    wind_set ( w_h2, WF_HSLSIZE, hslid_siz, 0, 0, 0 );
    pos = (int) ( (float)(lft) / ( (float)(col_cnt - cols_avail) ) * 1000;
    wind_set ( w_h2, WF_HSLIDE, pos, 0, 0, 0 );
}


open_vwork ()
{
    int i;

    handle = graf_handle ( &charw, &charh, &dun, &dun);
    for ( i=0; i<10; work_in[i++] = 1 );
    work_in[10] = 2;
    v_opnvwk ( work_in, &handle, work_out );
}


get_date ()
{
    int date, day, mnth, year;
    char d[3], m[3], y[4];

    date = Tgetdate ();
    day = date & 0x001f;
    mnth = (date >> 5) & 0x000f;
    year = ((date >> 9) & 0x007f) + 80;
    year = year % 100;
    sprintf ( d, "%d", day );
    sprintf ( m, "%d", mnth );
    sprintf ( y, "%d", year );
    if ( mnth < 10 ) {
       date_but[0] = '0';
       cur_date[0] = '0';
       strcpy ( &date_but[1], m );
       strcpy ( &cur_date[1], m );
    }
    else {
       strcpy ( date_but, m );
       strcpy ( cur_date, m );
    }
    date_but[2] = '/';
    if ( day < 10 ) {
       date_but[3] = '0';
       cur_date[2] = '0';
       strcpy ( &date_but[4], d );
       strcpy ( &cur_date[3], d );
    }
    else {
       strcpy ( &date_but[3], d );
       strcpy ( &cur_date[2], d );
    }
    date_but[5] = '/';
    if ( year < 10 ) {
       date_but[6] = '0';
       cur_date[4] = '0';
       strcpy ( &date_but[7], y );
       strcpy ( &cur_date[5], y );
    }
    else {
       strcpy ( &date_but[6], y );
       strcpy ( &cur_date[4], y );
    }
}

handle_keys ()
{}

handle_messages ()
{}

handle_button ()
{}
```

C-manship END

IAN'S QUEST

BY IAN CHADWICK

Is computer art really "art" in the traditional sense? That's a tough question. It's hard to define computer art, because it lacks solidity. After all, it's not really anywhere in a form commonly accepted in auctions. Even in a display it's only a transient visitor to a few RAM chips. And printed—well, what printer can capture all of the details of a screen?

Computer art certainly demands all of the same skills, learning, effort and talent of other art. But unlike all the rest of them, it has no permanent existence. On disk, it's merely a collection of magnetized oxide particles, prey to whatever cosmic mishap befalls disk data. Even if it remains stable, it is still more subject to change than any other form of art, since a single-bit adjustment creates a new piece. And since it's copyable, there is no such thing as an "original" as befits a great painter. Signed bitware, unchangeable artistic masters on disk? Not yet, anyway. Until then, "computer art" remains more of a craft than an art.

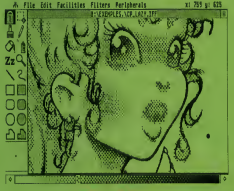
I'm no great shakes as an artist. Everything I draw seems to come out cockeyed. It's even worse with software. Doesn't matter what the drawing tool is or how many amazing features there are either. I *understand* art quite well: the process, perspective, viewpoint, the mechanics, all of that. It just doesn't translate itself well onto paper (or screen) for me.

However, that doesn't spoil my fun. Like most folks, I doodle, and given a computer paint program, I do it electronically. I've met others who do this quite proficiently and can whip up some rather impressive screens. The polite term for their work is "art," but I don't think the electronic Etch-A-Sketch stuff I produce deserves the distinction. I never managed to enchant any lady up to my garret to see my computer etchings, either—Susan was far too canny to fall for *that* line.

Part of my excuse for poor artistic performance may be valid for a lot of people with a higher level of talent, but who experience difficulty with computer drawing: The primary tool is inadequate, not the software, the mouse. I can do much better with a light pen and even produce something passably attractive with a digitizing tablet. But the mouse seems particularly unsuited for fine, freehand drawing. Maybe it's because it doesn't feel like a pen or a brush—or any of the vertically held tools we're taught to use. Or maybe it's because most mice have their rolling ball (which represents the pointer location as well) below the palm of your hand, rather than at the fingertips, where a pen or brush works.

Ever try to sketch something with a mouse? My dog could do better holding a crayon in his teeth! It sometimes seems that whatever I draw looks, at first, like a child's crude efforts. In order to compensate for the clumsiness of the mouse, I have to depend on software features such as zoom-level tools to correct straying lines and clean up nasty clumps of pixels; curve tools (Bezier and b-spline) to make lines look less like snail trails; line, ellipse, frame and so on for polygons. In art software, the more features, the better I like the program.

There are two sorts of art programs around: draw and paint. This column's about the latter; I'll do draw programs some other time. Basically the difference is that draw programs deal in "objects" and vectors (as most CAD programs do) and paint programs deal in pixels and screens. There is some overlap in functionality and features, but not enough for my taste.



MacPaint is the archetypal paint program from which many ST paint programs derive (as well as many PC paint programs). In it several important ideas were developed, as well as many icons, features and styles that are today accepted as standards.

DEGAS, one of the first such ST programs, and still one of the best, owes only some of its lineage to *MacPaint*. Author Tom Hudson took his own course, especially with *DEGAS Elite*. He prefers the dual command-screen/drawing-screen mode rather than putting all of his tools in the one screen with the work area, but the basic tools are the same. However, *DEGAS* is a bit long in the tooth these days and suffers from three main weaknesses: the rigid picture size (always exactly one screen large), the lack of curve and spline tools and the inability to cut nonrectangular blocks. These could be remedied, along with the addition of new tools and features, to bring *DEGAS Elite* back to the fore, but I personally don't believe Electronic Arts has sufficient interest in the ST market to make such a commitment.

I remember when I was working on the manual for *DEGAS*, back in the ST's Paleozoic. While learning the features, I spent hours and hours working on a "painting" of a wasp—lots of (for me, anyway) detail meticulously transcribed from a photograph, background scenery lovingly added in at zoom level. I was so proud of it that I offered it to Batteries Included (BI), the company that originally marketed *DEGAS*, to include on their disk as a sample picture, free of charge.

Some hope. Marty Herzog of BI smiled

condescendingly and showed me some of the samples they already had from people with what appeared to be an endless reservoir of talent in that area. As I watched the slide show of their efforts, my wasp became flat, uninviting, lifeless. I heaved a great sigh and went back to the writing, leaving the art to those who do it best.

So what? Well, being an artist is one of those hidden dreams that lurks deep inside me. Despite a propensity for drawing stick men and a sense of perspective only an alien could appreciate, I *love* to draw. Inside me lurks the heart of Picasso, Rembrandt, Da Vinci. That's why I can never avoid the lure of new art programs—in this case *HyperPaint* from Atari UK, *Touch-Up* from Migraph and two from Human Technologies in France: *Lazy Paint* and *Rough*. A real international sampling.

HyperPaint one might call Atari UK's answer to *DEGAS*. It is a competent paint program that adds a few features and enhancements to *DEGAS* but not everything that is needed. It supports GDOS and the manual has a rather good description of GDOS, ASSIGN.SYS and the installation process.

HyperPaint has Bezier curves, save/load .IMG files, ten workscreens (each of which can have its own palette), irregular block cut/copy/paste and a few other features, as well as the standard draw/paint/fill functions we know and expect. The interface is a little jagged and at times awkward. For example, if you choose Quit or click on the close box, you can choose to save the current workscreen (only!) or quit, but not cancel. It's an easy way to lose a lot of work. Also, you don't print from within the program—you do so through a separate output program, from which you can load or print a file.

Atari UK failed to provide any sample pictures in any format, a weakness for new users who'd rather tinker with predrawn work than attempt their own masterpieces before mastering the program. Facing a sparsity of paint programs, *HyperPaint* would be high on my list. But with so many others around—even some darn good public domain offerings—it doesn't offer enough novelty to recommend. It needs more chrome, more shine.

Touch-Up and *Lazy Paint* (which I believe is also known as *ZZ Paint*) are monochrome paint programs. This is unusual in itself: Publishers have tended to focus on the ST's color abilities and have treated monochrome as a poor cousin. Paint programs that have worked in mono have simply done the same things they do in

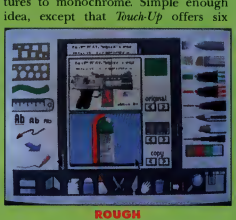
HYPERPAINT

color, without regard to any of the special needs and abilities of black and white. *Touch-Up* also works on a color monitor, in low and medium resolution, but only provides black and white "colors." *Lazy Paint* works strictly in mono.

Monochrome, by the way, isn't simply trading colors for higher resolution. It's a different metaphor completely. It's like the movies. A lot of movies shot in black-and white were so done because no one had color technology. On the other hand, a lot of directors *chose* black and white as their medium, despite the availability of color, because it offers special effects unattainable in color, and it also requires a different type of craftsmanship. Monochrome art is like that: to novices it looks like it offers less. But even if it's a lot more demanding, the results can be more satisfying.

In monochrome, it's much easier to do grey-scale gradations, and this should be an integral function in any mono-paint program. Many color paint programs provide a similar feature that permits automatic gradation or shifts, so it's reasonable to expect the same thing in mono-paint programs. If not done in actual grey scales, then it should be possible through patterns.

Touch-Up is one of those "Ferrari" programs: oodles of chrome, features and functions to complement the usual lot of commands. One example is the "map to black" option that translates color pictures to monochrome. Simple enough idea, except that *Touch-Up* offers six



ROUGH

methods of mapping, with variations. Migraph doesn't do anything by halves. Other extra features include B-spline and Bezier curves (with nice B-spline parameter settings); a .IMG file viewer; TIFF format support; a dynamic screen locator in zoom mode; measurements in inches; centimeters or pixels; fill patterns in 75, 150 and 300 dpi; GDOS support; many, many text, clip and display options; plus a whole lot more. A lot of thought went into this program.

There are some problems with *Touch-Up* however. One is the manual: First, it has no index—an inexcusably amateur oversight. Without an index, it is close to impossible to learn what *Touch-Up* can do without reading the manual over and over again. Second, the manual is poorly written, using the passive voice and many inexact terms or phrases. I'll give it a better rating than most Michiron manuals, but it could do with improvements.

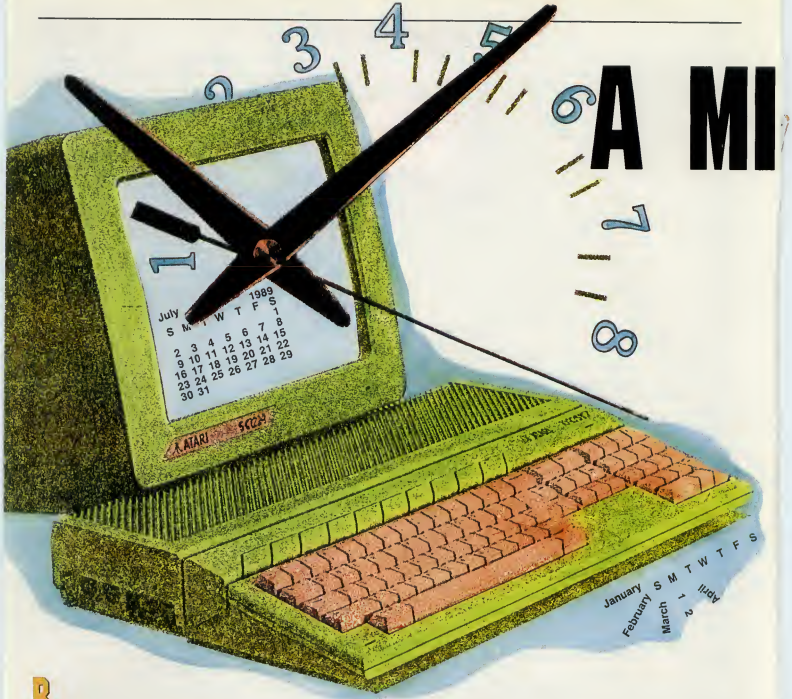
Other weaknesses include the lack of multiple magnification levels (only 2 x and 4 x), the slowness of the redraw and the glacial response time of some features (although a "lightning" mode is available, with fewer features), only one workscreen, the Undo key doesn't work and an awkward or poorly conceived user interface—for example, the nonrepetitive arrow buttons in the text-size dialog box and no grey scales. These faults combine to make what should be one of the best art programs around, merely a good program. Migraph should, and could, do better. I hope they heed this and upgrade both the minor program flaws and the manual.

Lazy Paint is a different type of program. It's a lot simpler, with fewer bells and whistles, but it has quite a few special items, such as three different grey-scales pattern bars, pattern search, portrait to landscape toggle, .PI3, .IFF, Postscript and .IMG file support, disk information tools, direct scanner support, user-definable filters for the clip box or the entire image and pointer coordinates displayed.

While the basic drawing tools in *Lazy Paint* are pretty much standard, the program is clean, smooth and easy to use. The few bits of chrome they chose to add are well thought-out and useful—not merely for show. It hasn't the muscle of *Touch-Up*, but it's still a nice program and one to consider for mono-only systems.

Rough (or *ZZ-Rough*) is France's answer to *Neochrome*. It's a low-resolution paint program that approaches the idea from a different angle. The program uses the

(to page 98)



Being able to measure time accurately in a software program can be helpful and sometimes essential. For real-time applications dealing with multiple events occurring outside the computer, it can be critical to know the exact time when those events occur. One example is in MIDI applications dealing with the time of occurrence of note-on or note-off commands. Unfortunately, it isn't always obvious just how to go about measuring time, especially if one isn't familiar with the details of the ST's hardware.

Real-time programs aren't the only ones that can make use of time-measurement techniques, though. Consider how nice it would be to have an easy way to program precise time delays or to measure the time required to execute a critical path in the new program you've just completed. Another useful software development tool might allow you to scatter markers through a troublesome program, giving a readout of the elapsed execution time between markers. In this article, we will develop these tools and provide the basis for others.

With one exception, the program functions are written in C, but they are well commented, which will allow you to transfer the ideas to the language of your choice. The millisecond timer's interrupt-service routine is the exception. It is written in 68000 assembly code because it is required in order to run in supervisor mode and must return using the privileged RTE instruction. All code was written using the Alcyon C Compiler included with the Atari Developer's Package, but should be adaptable to other C compilers with few if any changes.

LLISECOND TIMER

BY ROBERT H. OSNESS

What time is it?

The ST contains a real-time clock, but the clock's time values are accumulated in increments of two seconds—not a very useful resolution for events happening at electronic speeds! Unbeknownst to many of us, however, the designers of the ST cleverly included a 68901 multi-function peripheral (MFP) chip, which has four (count 'em) programmable timers. Most of these are allocated to various system functions, but one, Timer A, has been left for applications—that's us, folks!

When enabled, Timer A will generate an interrupt after a programmable number of cycles of its 2.4576 MHz input clock have elapsed, then reload itself for another timing cycle. Our program uses the ST's XBIOS functions *Xbtimer, Jenbint* and *Jdisint* to set up the timer and to enable or disable the associated interrupt. We program the timer to generate an interrupt every 1.0 millisecond, and use the interrupt to increment a memory word used as a counter. Reading the counter allows us to measure time directly in milliseconds.

It's possible to set up the timer to interrupt more frequently than once every millisecond, but it's not very desirable, because that could slow the ST in performing its other tasks. If greater timing resolution is necessary, there are better ways to get it as a future article will show.

The timer ISR

Where there are interrupts, there must be ISR's—that is, interrupt service routines. These are just like subroutines or function calls, except that a branch to the code they contain is executed when the corresponding interrupt occurs. The ISR then restores control to the normal program, wherever it was when the interrupt struck.

Listing 1 contains the assembly code for *__intr*, the millisecond timer's ISR. It is intentionally very short, to minimize system delays. All that happens is that the global variable *__timcnt* is incremented, and then the interrupt in-service bit is cleared before returning to the main program. The variable *__timcnt* contains the number of elapsed milliseconds since it was last initialized or cleared. It is declared as a long integer to allow time values greater than 65 seconds to be accumulated, which would be the limit for a regular 16-bit integer. Note that no register contents need to be saved in this ISR, because none are used.

The underscore must appear in front of the variables *__timcnt* and *__intr* for compatibility with the C compiler, which adds underscores to external variables. The variables *__timcnt* and *__intr* are declared as globals because the ISR is to be linked with the C code as a separately compiled module. The underscores are not used in the C source code.

The tools

Listing 2 contains the tools that expand the horizons of the millisecond timer into the real world. Let's skip over the main program segment for a moment and look at the subroutine functions.

Let's get started

The function *init__tmr( )* is used to set up the timer. *Jdisint* is an XBIOS function that disables Timer A's interrupt. *Xbtimer* is then called to set up the timer's prescale divider ratio, countdown value and ISR vector—that is, the ISR's address.

Jenabint is called to enable the timer's interrupt, and the millisecond timer is off and running! For those who want to know more of the hardware details, there are specifications for the timer in the 68901 data section of the Atari Developer's Package, beginning on page 984.

Delays, anyone?

Next, let's look at the function *wait__ms(ms)*. By passing a long-integer millisecond value to the function, we cause it to enter a timing loop, where it remains until the specified number of milliseconds have elapsed. Simple, isn't it?

There is a similar function for longer delays, *wait__sec(sec)*. It is the same, except that a long integer value for seconds of delay is passed.

Marking time

Now for a little more fun. To measure the execution time between two points in a program, we can use the function *calc__ms( )*. At the first of the two measurement points, we simply insert the statement *t1 = timcnt*, where *t1* is a long integer. Then, at the second measurement point, we insert a call to *calc__ms( )*, passing the *t1* and the string "since t1." The subroutine will measure the elapsed time, and will print the result to the screen as "Elapsed time = xxx milliseconds since t1," where the last two words are the string passed by the calling routine.

Successive calls can use additional declared variables *t2, t3*, etc. The time difference is also returned as an 16-bit integer, which may be used or ignored. Note that the variable *t2* used inside this subroutine is local and is not the same as the variable *t2* used in the main program.

The main thing

Now let's look at how *main( )* tests the timing subroutines. First we initialize the timer with a call to *init__timer( )*. Next comes a *while* loop, where we will remain until typing a "q" to quit. The first *printf* statement prints the counter value, which

should be 0 because the timer has just been started. Next is a 100-millisecond delay, then another *printf* statement to show how much time has elapsed to this point.

Now comes a combined test of the millisecond and second delay subroutines. The time is sampled in *t1*, then delays of ten seconds and 250 milliseconds are called, and finally *calc_ms()* is called to measure the elapsed time since *t1*.

Next is a test of a the execution time for a dummy *for* loop, which executes 9,000 times. The value of *t2* is used to indicate this result.

Last, another call to *calc_ms()* is used, with a zero in place of the identification string. This is just a variation in how the subroutine can be used if no id is needed.

The end of the loop has now been reached, and the user can enter a "q" to quit or repeat the sequence as many times as desired.

Let's build the program

Disk subscribers may now gloat, get out their program disks and skip the remainder of this section. Everyone else, please follow along!

Begin by typing in the timer ISR in Listing 1. Using the assembler disk in Drive A and your working disk in Drive B, assemble it using the batch file of Listing 3, BI2.BAT.

Now type in the C program TIMERS.C from Listing 2, and compile and link it using the batch file from Listing 4, BINT.BAT. Note the inclusion of "b:intr" in the link statement, to link in the object file INTO generated in the preceding step. Both of the batch files can be adapted to RAMdisk with minor modifications in order to speed assembly if desired.

Time to fly!

All that remains is to watch time fly. Click on TIMERS.PRG and we're off. In about 10.5 seconds, the first pass through the main loop will be complete, and we can look at the results.

At first glance, our 100-millisecond delay seems too long. Actually, the additional delay indicated on the second line of output is caused by the execution time of the *printf* function. This is demonstrated when we look at the next line. Exactly 10,250 milliseconds (that's 10.250 seconds) are indicated for the execution time of the ten-second and 250-millisecond delays.

The fourth line of output indicates that our dummy *for* loop takes 75 or 76 milliseconds to execute: about 8.3 microseconds for each pass through its loop.

The fifth line shows that the use of *calc_ms()* with a zero string parameter works as desired. It also indicates a delay of about 24 to 56 milliseconds due to the *printf* statement contained in the preceding call to *calc_ms()*.

Repeating the main program loop shows that the use of *printf* inserts a variable delay of 18 to 56 milliseconds. Though this is really not a long time in the human world, it can be of some importance in code segments that have to be fast.

Conclusion

In a future article, we will take a look at some ways to get even more timing accuracy. Until then, you may want to expand the program and try some code timing of your own. ■

Bob Osness, *who has been programming his ST for 2½ years, works as an electrical engineer for Boeing Aerospace in Kent, Washington. He and his wife, Georgia, spoil grandchildren as a hobby.*

MILLISECOND TIMER Listing 1: Assembly

```
.globl _timcnt
.globl _intr
.text
_intr:
addq.l #$1,_timcnt
andi.b #$DF,$fffa0f
rte
.data
.even
_timcnt: .dc.L 0
```

MILLISECOND TIMER Listing 2: C

```
#include <stdio.h>
#include <osbind.h>

extern long timcnt;             /* time counter */
extern int  intr();             /* ISR reference */

main()
{
     long t1, t2;               /* time values */
     int  i;                    /* loop count */
     char k;                    /* loop exit control character */

     k = 0;

     init_tmr();                /* initialize & start msec timer */

     while( k != 'q') {

          printf("Initial timcnt value is %D\n", timcnt);
          wait_ms(100L);             /* 100 msec delay */
          printf("After first delay, timcnt = %D\n", timcnt);

          t1 = timcnt;               /* time marker #1 */
          wait_sec(10L);             /* 10 second delay */
          wait_ms(250L);             /* 250 msec delay */

          calc_ms(t1,"since t1");    /* result should be 10250 msec */

          t2 = timcnt;               /* time marker #2 */
          for(i=0; i<9000; i++)      /* dummy code loop */
               k = k;
```

```
        calc_ms(t2,"since t2");      /* dummy code execution time */

        calc_ms(t2, 0L);             /* same call without id string */

        printf("To quit, type a 'q', otherwise ");
        printf("strike another key \n");
        k = Bconin(2);
    }

    Jdisint(13);                     /* disable timer a interrupt!
                                        required before exit!!     */

}

init_tmr()              /* Initialize Timer A */
{
    timcnt = 0;

    Jdisint(13);                     /* disable timer a interrupt */

    Xbtimer(0x00, 0x04, 49, intr);   /*   set up timer a:
                                       0x04 = prescale divide by 50
                                         49 = count down value
                                       intr = interrupt vector
                                              (ISR address)   */

    Jenabint(13);                    /* enable timer a interrupt */

}


wait_ms(ms)             /* Delay 'ms' milliseconds */
long ms;
{
    long delta, t1;

    delta = 0;
    t1 = timcnt;

    while(delta < ms) {
        delta = timcnt - t1;
    }
}

wait_sec(sec)           /* Delay 'sec' seconds */
long sec;
{
    long diff, t1;

    diff = 0;
    t1 = timcnt;

    while(diff < sec) {
        diff = (timcnt - t1) / 1000;    /* convert to seconds */
    }
}


calc_ms(t1, str)        /* Calculate elapsed time in milliseconds */
long t1;
char *str;
{
    long t2;

    t2 = timcnt;

    if(str)
        printf("Elapsed time = %D msec %s\n", t2 - t1, str);
    else
        printf("Elapsed time = %D msec \n", t2 - t1);

    return((int)(t2 - t1));
}
```

MILLISECOND TIMER
Listing 3:
Batch file

```
as68 -F b: -l -u X1.s
wait
```

MILLISECOND TIMER
Listing 4:
Batch file

```
cp68 X1.c X1.i
c068 X1.i X1.1 X1.2 X1.3 -f
rm X1.i
c168 X1.1 X1.2 X1.s
rm X1.1
rm X1.2
as68 -F b: -l -u X1.s
link68 [u] X1.68k=gemstart,X1,b:intr,gemlib,osbind,libf
relmod X1
rm X1.68K
wait
```

END

A MILLISECOND TIMER

The Game Cupboard

BY MARK E. NELSON

It seems that every suburban family room has one. Its stacks of taped-up cardboard boxes rattle noisily each time an evening of gaming begins. And four partial decks of cards rot for years in there, because those missing cards just *may* turn up. It's the game cupboard. Here's a game cupboard for your computer. And, as in any good game cupboard, there's something for everyone.

PHOTO • GARRY BROD

As in any good game cupboard, there's something for everyone.

Getting started

Double-click on CUPBOARD.PRG to begin (you must be in low resolution). You'll see the cupboard with a game sitting on each shelf. To play a game, just point to it and click the left mouse button. To leave the game cupboard, click in the EXIT box in the lower right-hand corner.

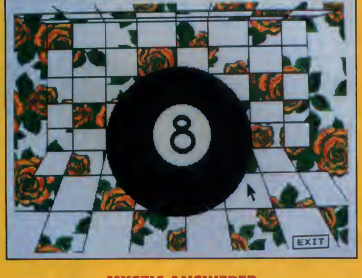
Naughts and Crosses

For the kids, there's a two-level tic-tac-toe game called *Naughts and Crosses*. Even adults may find it challenging to figure out how to beat the computer at this simple game. Click on EASIER or HARDER to set the level of play, and on EXIT when you're done playing. The computer will always let you move first.

Note: *The program listing at the end of this article is included only for those people who are interested in programming in Pascal. The program will not run without several picture files, which are included, along with the complete program, on this month's disk version or in the databases of the* ST-LOG *ST users' group on DELPHI. It is, of course, impossible to include the picture files within the pages of this magazine.*

NAUGHTS AND CROSSES

REPEAT



MYSTIC ANSWERER

Repeat

For older kids and adults, there's a memory game called *Repeat*. The computer will show a sequence of colored lights, and you must repeat it. The sequence will begin with one light, and continue to grow until you can't repeat it. Just click on a light to light it up. The highest score is saved to disk and displayed at the top of the screen to give you something to shoot for. Click on EXIT when you're done and ready to return to the cupboard.

Mystic Answerer

For everyone, there's an eight ball, called the *Mystic Answerer*, that knows the answers to all of life's questions. Just ask a question out loud, and then click the left mouse button to make the ball roll over and reveal the answer to your question. Of course be sure to ask your questions so that they require yes or no answers. Click on EXIT when you've had all of your questions answered.

Notes for programmers

I wrote *The Game Cupboard* using OSS Personal Pascal Version 2. The commented source code is shown at the end of this article and is also included on this month's disk. Feel free to use any portions of the code as long as you include the message: "Portions of this program were published in STLOG and were written by Mark E. Nelson" on the title screen and in the source code of your program. The sound routines are modified versions of those found on the OSS bulletin board. The modified routines are not totally disabled by key clicks as are the original routines.

The code should be fairly straightforward. I don't use many shortcuts or tricks. But if you do have any questions, leave me a message or mail on the Atari Connection BBS at 801-377-1617. ■

Mark Nelson *is a computer science student at Brigham Young University and the father of three boys, Drew, Steven and Aaron. He spends his free time coaching tee-ball, wrestling on the family room floor and watching* Sesame Street.

THE GAME CUPBOARD
Listing 1:
Pascal

THE GAME CUPBOARD

```
{ Title          The Game Cupboard          version 1.0
  Author         Mark E. Nelson
  Last Update    February 12, 1988
}

Program The_Game_Cupboard;
{                     Copyright (c) 1988 by ST-Log
     Portions of this code may be used freely as long as the title
     screen contains the message, "Portions of this program were
     published in ST-Log and were written by Mark E. Nelson."
}

{$P-}

{$I Gensubs.pas}
{$I Auxsubs.pas}

Const
  { constants for sound routines }
  cmd_write = 128;
  cmd_read  = 0;
  chana_lo  = 0;
  chana_hi  = 1;
  chana_vol = 8;
  chan_enable = 7;
  enable_sound=7;
  { musical notes for use with sound routines }
  A2 = 600;
  B2 = 530;
  C2 = 470;
  D2 = 437;
  E2 = 396;
  F2 = 350;
  G2 = 310;
  A3 = 300;
  B3 = 265;
  C3 = 235;
  D3 = 220;
  E3 = 198;
  F3 = 175;
  G3 = 157;
  A4 =150;
  A1 = 1200;
  B1 = 1060;
  C1 = 940;
  D1 = 880;
  E1 = 800;
  F1 = 700;
  G1 = 620;

  My_Brown = 7;
  Same = -1;

Type
  Channel = 0..2;       { for sound routines }

  lights_array = array[1..250] of integer;
  Screen_ptr = Record
                 case boolean of
                   False: (p:^Screen_L);  { Screen_L defined in AUXSUBS.PAS }
                   True: (l:Long_integer);
                 end;

Var
  Noise, Tone: boolean;       { for sound routines }
  Exit_Game, Button: boolean;
  Mouse_x, Mouse_y, num_answers: integer;
  Msg_Area:Message_Buffer;
  Eight_Scr, Cupboard_Scr, Naught_Scr, Repeat_Scr: Screen_type;
  Roll_Eight: array[1..5] of Screen_ptr;
  board: array[1..9] of integer;
  answers: array[1..100] of string;

Function Random(Max:Integer):Integer;
  FORWARD;

Function GetRez:Integer;
  XBIOS(4);

Function physbase: long_integer;
  XBIOS(2);

Function logbase: long_integer;
  XBIOS(3);
Procedure setscreen(logadr, physadr: long_integer; res: integer);
  XBIOS(5);

Procedure Sav_Scn(Sav_To: Screen_Ptr);
  var Sav_From: Screen_Ptr;
  begin
    Sav_From.l:=Physbase;
    Sav_To.p^:=Sav_From.p^
  end; { of Proc Sav_Scn }

Procedure Rest_Scn(Sav_From: Screen_Ptr);
  var Sav_To: Screen_Ptr;
  begin
    Sav_To.l:=Physbase;
    Sav_To.p^:=Sav_From.p^
  end; { of Proc Rest_Scn }

Procedure Str(value:integer; var number:string);
{ This Procedure converts an integer value to a string of digits }

Var
  temp,I:Integer;
  Gotit:boolean;
begin
  Gotit:=false;
  Number:='';
```

THE GAME CUPBOARD

```
 If value<0 then begin
   Number:='-';
   Value:=abs(value);
  end;
 If value=0 then number:='0';
 For i:=4 downto 0 do begin
  temp:=value div round(power(ten[i]));
  If temp>0 then Gotit:=true;
  If Gotit then number:=concat(number,chr(ord(48)+temp));
  Value:=Value- temp*round(power(ten[i]))
  end
 end;{ of Procedure Str }


Procedure Main_Event(Time:Long_Integer);
 FORWARD;

Procedure Box_Cmd(cmd,i1,i2,i3,i4,i5,i6,i7,i8:integer; Var out1,out2:integer);
 Var
  int_in:Int_in_parms;
  int_out:Int_out_parms;
  Addr_in:Addr_in_parms;
  Addr_out:Addr_out_parms;

Begin
 int_in[0]:=i1;
 int_in[1]:=i2;
 int_in[2]:=i3;
 int_in[3]:=i4;
 int_in[4]:=i5;
 int_in[5]:=i6;
 int_in[6]:=i7;
 int_in[7]:=i8;
 AES_Call(cmd,int_in,Int_out,addr_in,addr_out);
 out1:=int_out[1];
 out2:=int_out[2]
End; { of procedure Box_Cmd }

Procedure Grow_Shrink(cmd,small_x, small_y, small_w, small_h,
                      big_x, big_y, big_w, big_h:integer );

 Var
  int_in:Int_in_parms;
  int_out:Int_out_parms;
  addr_in:addr_in_parms;
  addr_out:addr_out_parms;

 Begin
  Int_in[0]:=small_x;
  Int_in[1]:=small_y;
  Int_in[2]:=small_w;
  Int_in[3]:=small_h;
  Int_in[4]:=big_x;
  Int_in[5]:=big_y;
  Int_in[6]:=big_w;
  Int_in[7]:=big_h;
  AES_Call(cmd, int_in, Int_out, addr_in, addr_out)
End; { of Procedure Grow_Shrink }
Procedure Grow_Box(small_x, small_y, small_w, small_h,
                   big_x, big_y, big_w, big_h:integer);
 Begin
  Grow_Shrink(73, small_x, small_y, small_w, small_h,
              big_x, big_y, big_w, big_h)
 End; { of Prodedure Grow_Box }

Procedure Shrink_Box(big_x, big_y, big_w, big_h,
                     small_x, small_y, small_w, small_h:integer);
 Begin
  Grow_Shrink(74, small_x, small_y, small_w, small_h,
              big_x, big_y, big_w, big_h)

 End; { of procedure Shrink_Box }

Procedure Move_Box(x,y,w,h,new_x,New_y:Integer);
begin
 Grow_Shrink(72,w,h,x,y,new_x,new_y,0,0)
end; { of procedure Move_Box }

Procedure Wait( Seconds: Long_Integer);
Var Message:Message_Buffer;
    d:integer;
Begin
 Seconds:=Seconds*1000;
 Main_Event(Seconds)
End; { of procedure Wait }

Function gia_read (data, register:integer):integer;
 XBIOS (28);

Procedure gia_write (data, register:integer);
 XBIOS (28);

Procedure Sound_init;
 Var
  Port_state:integer;

 Begin
  port_state:=gia_read(0,chan_enable+cmd_read);
  gia_write(port_state&(~enable_sound),chan_enable+cmd_write);
  {Next three lines added by me}
  port_state:=gia_read(8,7+cmd_read);
  gia_write(port_state | 56,7+cmd_write);
  port_state:=gia_read(8,7+cmd_read);
  If Noise then gia_write(port_state & (~56),7+cmd_write)
           else gia_write(port_state | 56,7+cmd_write);
  port_state:=gia_read(8,7+cmd_read);
  If Tone then gia_write(port_state & (~7),7+cmd_write)
          else gia_write(port_state | 7,7+cmd_write)

 End; { Of Procedure Sound_init }

Procedure Sound(ch:channel; pitch:integer; vol:integer);
 Begin
```

```
  Sound_Init;
  gia_write(vol,chana_vol+ch+cmd_write);
  gia_write(pitch&$FF, chana_lo+ch*2+cmd_write);
  gia_write(shr(pitch,8),chana_hi+ch*2+cmd_write);
  gia_write(pitch&$FF,6+cmd_write)
 end; { Of Procedure Sound }

Procedure Sound_off;

 Var
  port_state:integer;

 Begin
  Sound(0,0,0);
  Sound(1,0,0);
  Sound(2,0,0);
  port_state:=gia_read(8,chan_enable+cmd_read);
  gia_write(port_state|enable_sound, chan_enable+cmd_write)
 end; { Of Procedure Sound_Off }

Procedure Wait_Sound(Time:integer);
var x,y,j:Integer;
Begin
 For J:=1 to 100 do For x:=1 to Time do y:=x*100-J+14*2 div 100
end; { Of Procedure Wait_Sound }

Procedure Wrong_Sound;
 Var
  Count:Integer;
 Begin
  Tone!=True;Noise!=False;
  For Count:=1 to 2 do begin
    Sound(1,2260,15);
    Sound(2,2538,15);
    Sound(0,2260,15);
    Wait_Sound(30);
    Sound_Off;
    Wait_Sound(4)
   end
 end;  { Of procedure Wrong_Sound }

Procedure Happy_Sound;
  Var
   Volume,Note:Integer;
  Begin
   Volume:=12;
   Tone!=True;Noise!=False;
   For note:=1 to 3 do begin
     Sound(0,a2,volume);
     Sound(1,c2,volume);
     Sound(2,e2,volume);
     If Note=1 then Wait_Sound (60) else Wait_Sound(35);
     Sound_off;Wait_Sound(1)
    end;
  Sound(0,a4,volume);
  Sound(1,c3,volume);
  Sound(2,e3,volume);
  Wait_Sound(200);
  For Note:=1 to volume-5 do begin
     Sound(0,a4,volume-Note);
     Sound(1,c3,volume-Note);
     Sound(2,e3,volume-Note);
     Wait_Sound(2)
    end;
   Sound_off
 end; { Of Procedure Happy_Sound }

Procedure Comp_moved_Sound;
var volume: integer;
begin
  tone:=true; noise:=false;
  volume:=12;
  Sound(0,a4,volume);
  Sound(1,a3,volume);
  Sound(2,a2,volume);
  wait_sound(20);
  Sound(0,a3,volume);
  Sound(1,a2,volume);
  Sound(2,a1,volume);
  wait_sound(20);
  Sound_Off;
end; { of Procedure Comp_moved_Sound }

Procedure Hum_moved_sound;
var volume: integer;
begin
  tone:=TRUE; noise:=FALSE;
  volume:=12;
  Sound(0,a3,volume);
  sound(1,a2,volume);
  sound(2,a1,volume);
  wait_sound(20);
  sound(0,a4,volume);
  sound(1,a3,volume);
  sound(2,a2,volume);
  wait_sound(20);
  sound_off;
end; { of procedure Hum_moved_sound }

Procedure Lights_Sound(Light, Length: integer);
var volume: integer;
begin
  tone:= TRUE; noise:= FALSE;
  volume:=12;
  Case light of
     1: begin
          Sound(0,a3,volume);
          Sound(1,a2,volume);
          Sound(2,a1,volume);
        end;
     2: begin
          Sound(0,b1,volume);
          Sound(1,b2,volume);
          Sound(2,b1,volume);
        end
     3: begin
          Sound(0,c3,volume);
          Sound(1,c2,volume);
          Sound(2,c1,volume);
        end;
     4: begin
          Sound(0,d1,volume);
          Sound(1,d2,volume);
          Sound(2,d1,volume);
        end;
  end;
  Wait_Sound(Length);
  Sound_off;
end; { of procedure lights_sound }

Procedure Correct_Sound;
 Var
  Volume,Note:Integer;
 Begin
  Volume:=11;
  Tone:=True;Noise:=False;
  Note:=1;
  Repeat
    Sound(1,993-Note,Volume);
    Sound(2,1001-Note,Volume);
    Sound(0,997-Note,Volume);
    Note:=Note+15;
  Until Note>1000;
  Sound_off
 end; { Of Procedure Correct_Sound }

Function XB_Rnd:Long_Integer;
 XBIOS(17);

Function Random; (Declared prior with FORWARD )
 var i:Integer;
 Begin
  i:=Abs(Int(XB_Rnd) Mod (Max+1));
  If (i<1) OR (i>Max) then i:=Random(Max);
  Random:=i
 End; { of Function Random }

Procedure Get_Out;
{ exits the program unconditionally }
 Begin
  Exit_Gem;
  Halt
 End; { of procedure Get_out }

Procedure Main_Event; { declared prior with FORWARD }
 Var d,event,bstate:integer;    { d is a dummy variable }

 begin
    event:=Get_Event(E_Button|E_Timer,1,1,1,Time,
                     False,0,0,0,0,False,0,0,0,0,
                     msg_area,d,d,bstate,Mouse_x,Mouse_y,d);

If (Event & E_Button) <> 0 ( Then a button has been pressed )
   then
     button:= TRUE

  else                  ( a timer event has occured )
     button:= FALSE;

end; { of procedure Main_Event }

Procedure Get_Button;
begin
   Repeat
     Main_Event(100)                ( wait for mouse button input )
   Until button;
end; { of Procedure Get_Button }

Procedure Init_Screens;
{ set up the eight ball screens }
var i: integer;
    log: long_integer;
begin
  hide_mouse;
  set_clip(0,0,320,200);
  log:=logbase;
  for i:= 1 to 5 do begin
    new(Roll_Eight[i].p);
```

THE GAME CUPBOARD

```
    Roll_Eight[i].p^:=Eight_Scr.pic;  { Each gets a copy of 8 ball screen }
  end;
  setscreen(Roll_Eight[1].l, Same, Same); { work on screen 1 }
  paint_color(black);
  paint_style(1);
  paint_oval(150,110,67,67);              { 8 ball itself }
  paint_Color(white);
  paint_oval(150,125,20,20);              { white circle in 8 ball }
  paint_Color(black);
  for i:=1 to 5 do begin
    frame_oval(150,116,(4+i),(4+i));    { draw top circle of 8 }
    frame_oval(150,131,(8+i),(7+i));    { bottom circle of 8 }
  end;
  setscreen(Roll_Eight[2].l, Same, Same); { work on screen 2 }
  paint_color(black);
  paint_oval(150,130,95,95);                { 8 ball itself }
  paint_color(white);
  paint_oval(150,177,35,35);              { white circle in 8 ball }
  paint_color(black);
  for i:=1 to 8 do begin
    frame_oval(150,168,(4+i),(4+i));
    frame_oval(150,189,(5+i),(7+i));
  end;
  setscreen(Roll_Eight[3].l, Same, Same); { work on Screen 3 }
  paint_oval(150,140,110,110);              { 8 ball itself }

  setscreen(Roll_Eight[4].l, Same, Same); { work on screen 4 }
  paint_oval(150,150,125,125);              { 8 ball itself }
  paint_color(11);  { dark blue }
  paint_round_rect(110,60,100,15);

  setscreen(Roll_Eight[5].l, Same, Same); { work on screen 5 }
  paint_color(black);
  paint_oval(150,160,140,140);              { 8 ball itself }
  paint_color(11);  { dark blue }
  paint_round_rect(100,110,120,30);

  setscreen(log, Same, Same);
  show_mouse;
end; { of procedure init_screens }

Procedure Initialize;
  Var
  Rez,x:integer;
  f: file of text;
begin
  Hide_Mouse;
  Exit_Game:= FALSE;
  Rez:=GetRez;
  if Rez=1 then begin
    x:=Do_Alert ('[0][  |  Switch to Low Resolution      |  ][  OK  ]',1);
    Get_Out
    end;
  Clear_Screen;
  x:=Read_Scrn('CUPBOARD.P11', Cupboard_Scr);   { Read picture screens }
  ScrRest(Cupboard_Scr);
  Set_Mouse(M_Bee);
  Show_Mouse;
  x:=Read_Scrn('EIGHT.P11', Eight_Scr);
  x:=Read_Scrn('MOUGHT.P11',Mought_Scr);
  x:=Read_Scrn('REPEAT.P11',Repeat_Scr);
  Reset(f,'ANSWERS.EIT');
  num_answers:=0;
  While ((NOT EOF(f)) AND (num_answers<100)) do begin
    num_answers:=num_answers+1;
    Readln(f,answers[num_answers]); { read 8 ball answers }
  end;
  close(f);
  Init_Screens;
  Set_Mouse(M_Arrow);
end; { of Procedure Initialize }

{*************************************************************************
************************** Repeat Procedures ****************************
*************************************************************************}

Procedure light(which_light, length: integer);
Const
  w = 70;
  h = 43;

begin
  hide_mouse;
  case which_light of
    1: begin
         Paint_Color(6);
         Paint_Rect(77,32,w,h);
         lights_sound(which_light, length);
         Paint_Color(2);
         Paint_Rect(77,32,w,h);
       end;
    2: begin
         Paint_Color(8);
         Paint_Rect(172,32,w,h);
         lights_sound(which_light, length);
         Paint_Color(14);
         Paint_Rect(172,32,w,h);
       end;
    3: begin
         Paint_Color(8);
         Paint_Rect(77,108,w,h);
         lights_sound(which_light, length);
         Paint_Color(4);
         Paint_Rect(77,108,w,h);
       end;
    4: begin
         paint_color(10);
         Paint_Rect(172,108,w,h);
         lights_sound(which_light, length);
         paint_Color(11);
         Paint_Rect(172,108,w,h);
       end;
```

```
  end;
  show_mouse;
end; { of procedure light }

Procedure Show_Lights(var lights: lights_array; var count: integer);
{ Shows the previous moves and adds another }
var i, length: integer;
begin
  wait_sound(200);                          { pause a moment }
  count:=count+1;                           { inc count }
  if (count< 6) then length:= 100           { set sound length }
    else if (count<11) then length:=75
    else if (count<16) then length:=50
    else if (count<21) then length:=40
    else if (count<26) then length:=35
    else length:=30;
  lights[count]:= random(4);                { add a new light }
  for i:= 1 to (count-1) do begin           { show the lights }
    light(lights[i], length);
    wait_sound((length div 10));
    end;
  light(lights[count], 100);  { show last light a tad longer }
end; { of Procedure Show_Lights }

Procedure Get_A_Light(lights: lights_array; which_light: integer;
                      var Wrong, Exit_Repeat: boolean);
var x,y, selected,i: integer;
    OK: boolean;
begin
  OK:= FALSE; selected:=0;
  Repeat
  get_button;
  x:= mouse_x;  y:= mouse_y;
  if ((x)=72) AND (x(=146)) then      { Left side }
    if ((y)=32) AND (y(=74)) then     { top Left  }
      selected:=1 else
    if ((y)=100) AND (y(=142)) then  { bottom left }
      selected:=3 else
  else
  if ((x)=172) AND (x(=241)) then     { Right Side }
    if ((y)=32) AND (y(=74)) then     { top right  }
      selected:=2 else
    if ((y)=100) AND (y(=142)) then  { bottom right }
      selected:=4;
  if ((x)=141) AND (x(=178) AND (y)=82) AND (y(=92))
    then Exit_Repeat:=TRUE;
  OK:= (selected()0)
  Until (OK OR Exit_Repeat);
  light(selected, 50);
  if (selected=which_light) then Wrong:=FALSE
    else Wrong:=TRUE;
  if (Wrong AND (NOT Exit_Repeat)) then begin
    wrong_sound;
    for i:= 1 to 5 do
      light(which_light,15);
    end;
end; { of Procedure Get_A_Light }

Procedure Get_Lights(lights: lights_array; count: integer;
                     var game_over, Exit_Repeat: boolean);
{ gets human input and checks for validity. }
var i: integer;
begin
  i:=1; Game_Over:= FALSE;
  Repeat
    Get_A_Light(lights,lights[i],Game_Over,Exit_Repeat);
  i:=i+1
  Until (Game_Over OR (i)count) or Exit_Repeat);
end; { of procedure get_lights }

Procedure Update_Score(var hi, score: integer; var New_hi: boolean);
{ updates the hi score and the reg score }
var st: string;
begin
  Hide_mouse;
    Str(score,st);
    draw_String(24,30,st);
    If (score)hi) then begin
      New_hi:=TRUE;
      hi:=score;
      str(hi,st);
      draw_string(272,30,st);
    end;
  Show_mouse;
end; { of procedure update_score }

Procedure Repeat_Game;
var
  lights: lights_array;
  Count, i, hi: integer;
  Exit_Repeat, Game_Over, New_hi: boolean;
  st: string;
  f: file of integer;

begin
  Grow_Box(118, 88, 76, 30, 0, 0, 320, 200);
  Hide_Mouse;
  ScrRest(Repeat_Scr);
  Show_mouse;
  Exit_Repeat:= FALSE; New_hi:=FALSE;

  IO_Check(FALSE);
  Reset(f,'Repeat.hi');
  If (IO_Result)=0) then begin
    Read(f,hi);
    close(f);
    end
  else hi:=0;

  Repeat
    Hide_Mouse;
    Draw_String(52,148,'Click mouse button to play.');
```

THE GAME CUPBOARD

```
    Show_Mouse;
    get_button;
    Hide_Mouse;
    ScrRest(Repeat_Scr);
    str(hi,st);
    Draw_String(272,30,st);
    Draw_String(24,30,'0    ');
    Show_Mouse;
    Count:=0;  Game_Over:= FALSE;
    Repeat
      Show_lights(lights, count);
      Get_Lights(lights,count,Game_Over,Exit_Repeat);
      If NOT Game_Over then Update_Score(hi, count, New_hi);
    Until (Game_Over OR Exit_Repeat)
  Until Exit_Repeat;

  If New_hi then begin          ( save hi score to disk )
    Rewrite(f,'Repeat.hi');
    Write(f,hi);
    close(f);
  end;

  hide_Mouse;
  ScrRest(Cupboard_Scr);
  Shrink_Box(0,0,320,200,118,88,76,30);
  Show_Mouse;
end; { of Procedure Repeat_Game }

{**********************************************************************
********************* Eight Ball Game Procedures **********************
**********************************************************************}

Procedure Show_Answer;
var i: integer;
    temp_scr: screen_ptr;
    ans,temp1,temp2: string;
begin
  Hide_Mouse;
  new(temp_scr,p);
  Sav_Scn(temp_scr);
  for i:=1 to 5 do begin                ( roll the ball )
    Rest_Scn(Roll_Eight[i]);
    wait_sound(5);
  end;

  i:= random(num_answers);              ( randomly select an answer )
  ans:=answers[i];

  if (length(ans)>14) then              ( parse ans into two strings )
    begin
      i:=15;
      while (i>1) AND (ans[i]<>' ') do i:=i-1;  ( search for space )
      if i<=1 then              ( no spaces found )
        begin
          temp1:=ans;
          temp2:='';
        end
      else                      ( space WAS found )
        begin
          temp1:=copy(ans,1,i-1);   ( first i elements to temp1 )
          temp2:=copy(ans,i+1,length(ans)-i)); ( rest to temp2 )
        end;
    end
  else
    begin
      temp1:=ans;
      temp2:='';
    end;

  Text_color(13);
  Set_Color(13,0,0,1000);
  Draw_mode(2);                         ( draw the answer )
  if temp2<>'' then begin
  {Draw_String(160-(4*length(temp1)), 122, temp1);
  {Draw_String(160-(4*length(temp2)), 132, temp2);
  end else
    Draw_String(160-(4*length(temp1)), 127, temp1);
  Show_Mouse;
  Draw_mode(1);
  Text_Color(black);

  for i:=0 to 1000 do                   ( fade words in )
    set_color(13,i,i,1000);

  get_button;                           ( wait for a button event )

  Hide_mouse;
  for i:=5 downto 1 do begin      ( unroll the ball )
    Rest_Scn(Roll_Eight[i]);
    wait_Sound(5);
  end;
  Rest_Scn(temp_scr);
  Show_Mouse;
  dispose(temp_scr,p);
end; { of procedure Show_answer }

Procedure Answer;
var Exit_Answer: boolean;
    x, y: integer;
begin
  Exit_Answer:= FALSE;
  Repeat
    get_button;
    x:= mouse_x; y:= mouse_y;
    if ((x>=263) AND (x<=299) AND (y>=189) AND (y<=199))
      then Exit_Answer:= TRUE
      else Show_Answer;
  Until Exit_Answer;
end; { of Procedure Answer }

Procedure Eight_Ball;
begin
```

```
  Grow_Box(118, 124, 76, 30, 0, 0, 320, 200);
  Hide_Mouse;
  ScrRest(Eight_Scr);
  Show_Mouse;
  Answer;
  Hide_Mouse;
  ScrRest(Cupboard_Scr);
  Shrink_Box(0,0,320,200,118,124,76,30);
  Show_mouse;
end; { of Procedure Eight_Ball }

{*****************************************************************
*********************** Noughts Procedures ***********************
*****************************************************************}

Procedure Show_Move(Letter: string; move: integer);
{ Shows the letter in position held by move }
begin
  hide_mouse;
  Text_Height(20);
  case move of                        { place an 'x' or an 'o' }
    1: Draw_String(103,34,Letter);
    2: Draw_String(146,34,Letter);
    3: Draw_String(191,34,Letter);
    4: Draw_String(103,67,Letter);
    5: Draw_String(146,67,Letter);
    6: Draw_String(191,67,Letter);
    7: Draw_String(103,99,Letter);
    8: Draw_String(146,99,Letter);
    9: Draw_String(191,99,Letter);
  end;
  Hum_moved_sound;
  Text_Height(6);
  Show_mouse;
end; { of Procedure Show_Move }

Procedure Get_Move(var exit_noughts, harder: boolean);
{  Gets a human move, shows it, and stores it.
   exit_noughts becomes true if exit is clicked rather than a move.
   Will not allow a move to a space that is not vacant.  }
Const
  Exit_Top      = 189;
  Exit_Bottom   = 199;
  Exit_Left     = 135;
  Exit_Right    = 172;
  Top           = 10;
  Top_Mid       = 44;
  Bottom_mid    = 77;
  Bottom        = 112;
  Left          = 81;
  Left_mid      = 127;
  Right_mid     = 172;
  Right         = 225;
  Easy_Left     = 16;
  Easy_Right    = 64;
  Hard_Left     = 251;
  Hard_Right    = 302;

var
  done, Left_Col, Mid_Col, Right_Col, Top_Row, Mid_Row, Bot_Row: boolean;
  x, y, move: integer;

begin
  Exit_Noughts:= FALSE; Left_Col:= FALSE; Mid_Col:= FALSE;
  Right_Col:= FALSE; Top_Row:= FALSE; Mid_Row:= FALSE;
  Bot_Row:= FALSE; done:= FALSE;
  Repeat
    Main_Event(100);
    If Button then begin
      x:= mouse_x;  y:= mouse_y;
      If ((x >= Exit_Left) AND (x <= Exit_Right) AND    { Check for Exit }
          (y >= Exit_Top)  AND (y <= Exit_Bottom))
        then Exit_Noughts:= TRUE;

      If ((x >= Easy_Left) AND (x <= Easy_Right) AND
          (y >= Exit_Top)  AND (y <= Exit_Bottom))
        then begin
          Hide_Mouse;
          Harder:= FALSE;
          Line_Color(Black);
          Line(Easy_Left,Exit_Bottom,Easy_Right,Exit_Bottom);
          Line(Easy_Left,Exit_Top,Easy_Right,Exit_Top);
          Line_Color(My_Brown);
          Line(Hard_Left,Exit_Bottom,Hard_Right,Exit_Bottom);
          Line(Hard_Left,Exit_Top,Hard_Right,Exit_Top);
          show_mouse;
        end;

      If ((x >= Hard_Left) AND (x <= Hard_Right) AND
          (y >= Exit_Top)  AND (y <= Exit_Bottom))
        then begin
        hide_mouse;
        Harder:= TRUE;
        Line_Color(My_Brown);
        Line(Easy_Left,Exit_Bottom,Easy_Right,Exit_Bottom);
        Line(Easy_Left,Exit_Top,Easy_Right,Exit_Top);
        Line_Color(Black);
        Line(Hard_Left,Exit_Bottom,Hard_Right,Exit_Bottom);
        Line(Hard_Left,Exit_Top,Hard_Right,Exit_Top);
        show_mouse;
      end;

     If ((x >= Left) AND (x <= Left_mid)) then Left_Col:= TRUE else
     If ((x >= Left_mid) AND (x <= Right_mid)) then Mid_Col:= TRUE else
     If ((x >= Right_mid) AND (x <= Right)) then Right_Col:= TRUE;
     if ((y >=Top) AND (y <= Top_mid)) then Top_Row:=TRUE else
     if ((y >=Top_mid) AND (y <= Bottom_mid)) then Mid_Row:= TRUE;
     if ((y >=Bottom_mid) AND (y <= bottom)) then Bot_Row:= TRUE;


     If (Left_Col AND Top_Row AND (board[1]=0)) then begin
         Show_Move('X',1);
```

THE GAME CUPBOARD

PROGRAM LISTINGS

THE GAME CUPBOARD

```
                Board[1]:= 1;
                done:= TRUE;
                end else
            If (Left_Col AND Mid_Row AND (board[4]=0)) then begin
                Show_Move('X',4);
                board[4]:=1;
                done:= TRUE;
                end else
            If (Left_Col AND Bot_Row AND (board[7]=0)) then begin
                Show_Move('X',7);
                board[7]:=1;
                done:= TRUE;
                end else
            If (Mid_Col AND Top_Row AND (board[2]=0)) then begin
                Show_Move('X',2);
                board[2]:= 1;
                done:= TRUE;
                end else
            If (Mid_Col AND Mid_Row AND (board[5]=0)) then begin
                Show_Move('X',5);
                board[5]:=1;
                done:= TRUE;
                end else
            If (Mid_Col AND Bot_Row AND (board[8]=0)) then begin
                Show_move('X',8);
                board[8]:= 1;
                done:= TRUE;
                end else
            If (Right_Col AND Top_Row AND (board[3]=0)) then begin
                Show_move('X',3);
                board[3]:=1;
                done:= TRUE;
                end else
            If (Right_Col AND Mid_Row AND (board[6]=0)) then begin
                Show_move('X',6);
                board[6]:=1;
                done:= TRUE;
                end else
            If (Right_Col AND Bot_Row AND (board[9]=0)) then begin
                Show_move('X',9);
                board[9]:= 1;
                done:= TRUE;
                end;

    end;
  Until done OR Exit_Noughts;
end; { of Function Get_Move }

Procedure Comp_Move( Move: integer );
{ Marks the computer move at board[move) and shows it on screen. }
begin
  board[move]:= 2;
  Show_Move('O',move);
  comp_moved_sound;
end; { of Procedure Comp_Move }

Function All_2(x,y,z: integer): boolean;
{ returns true if board(x) = board(z) are all 2's }
begin
  All_2:= (board[x]=2) AND (board[y]=2) AND (board[z]=2);
end; { of Function All_2 }

Function All_1(x,y,z: integer): boolean;
{ returns true if board(x) = board(z) are all 1's }
begin
  All_1:= (board[x]=1) AND (board[y]=1) AND (board[z]=1);
end; { of Function All_2 }

Function Comp_Wins: boolean;
{ returns true if computer wins the game }
begin
  Comp_Wins:= ( All_2(1,2,3) OR All_2(1,4,7) OR All_2(1,5,9) OR
                All_2(3,5,7) OR All_2(3,6,9) OR All_2(7,8,9) OR
                All_2(2,5,8) OR All_2(4,5,6) );
end; { of Function Comp_Wins }

Function Human_Wins: boolean;
{ returns true if human wins the game }
begin
  Human_Wins:= ( All_1(1,2,3) OR All_1(1,4,7) OR All_1(1,5,9) OR
                 All_1(3,5,7) OR All_1(3,6,9) OR All_1(7,8,9) OR
                 All_1(2,5,8) OR All_1(4,5,6) );
end; { of Function Human_Wins }

Function Two_2(x,y,z: integer; var vacant: integer): boolean;
{ returns true if two of board(x)= board(y) are 2's and board(vacant) = 0 }
var temp: boolean;
begin
  temp:= FALSE;
  If ((board[x]=2) AND (board[y]=2) AND (board[z]=0))
     then begin
       vacant:=z;
       temp:= TRUE;
     end
  else if ((board[x]=2) AND (board[z]=2) AND (board[y]=0))
    then begin
      vacant:=y;
      temp:= TRUE;
    end
  else if ((board[y]=2) AND (board[z]=2) AND (board[x]=0))
    then begin
      vacant:= x;
      temp:= TRUE;
    end;
  Two_2:= temp;
end; { of Function Two_2 }

Function Two_1(x,y,z: integer; var vacant: integer): boolean;
{ returns true if two of board(x)= board(y) are 1's and board(vacant) = 0 }
var temp: boolean;
begin
  temp:= FALSE;
```

THE GAME CUPBOARD

```
  If ((board[x]=1) AND (board[y]=1) AND (board[z]=0))
    then begin
      vacant:=z;
      temp:= TRUE;
    end
  else if ((board[x]=1) AND (board[z]=1) AND (board[y]=0))
    then begin
      vacant:=y;
      temp:= TRUE;
    end
  else if ((board[y]=1) AND (board[z]=1) AND (board[x]=0))
    then begin
      vacant:= x;
      temp:= TRUE;
    end;
  Two_1:= temp;
end; { of Function Two_1 }

Function Win_Space(var vacant: integer): boolean;
{ returns true if there's a space that can result in the computer
  winning the game.  Vacant returns the space.  }
var temp: boolean;
begin
  temp:= ( Two_2(1,2,3,vacant) OR Two_2(1,4,7,vacant)
                OR Two_2(1,5,9,vacant) OR Two_2(3,5,7,vacant)
                OR Two_2(3,6,9,vacant) OR Two_2(7,8,9,vacant)
                OR Two_2(2,5,8,vacant) OR Two_2(4,5,6,vacant) );

  If temp then
    if Two_2(1,2,3,vacant) then
    else if Two_2(1,4,7,vacant) then
    else if Two_2(1,5,9,vacant) then
    else if Two_2(3,5,7,vacant) then
    else if Two_2(3,6,9,vacant) then
    else if Two_2(7,8,9,vacant) then
    else if Two_2(2,5,8,vacant) then
    else if Two_2(4,5,6,vacant) then;
  Win_Space:=temp;
end; { of Function Win_Space }

Function Block_Needed(var vacant: integer): boolean;
{ returns true if the computer needs to block a win by the human.
  vacant returns the space that must be moved into to block.  }
var temp: boolean;
begin
  temp:= ( Two_1(1,2,3,vacant) OR Two_1(1,4,7,vacant) OR
                Two_1(1,5,9,vacant) OR Two_1(3,5,7,vacant) OR
                Two_1(3,6,9,vacant) OR Two_1(7,8,9,vacant) OR
                Two_1(2,5,8,vacant) OR Two_1(4,5,6,vacant) );

  If temp then
    if Two_1(1,2,3,vacant) then
    else if Two_1(1,4,7,vacant) then
    else if Two_1(1,5,9,vacant) then
    else if Two_1(3,5,7,vacant) then
    else if Two_1(3,6,9,vacant) then
    else if Two_1(7,8,9,vacant) then
    else if Two_1(2,5,8,vacant) then
    else if Two_1(4,5,6,vacant) then;
  Block_Needed:=temp;
end; { of Function Block_Needed }

Function Tie_Game: boolean;
{ returns true if all spaces on board are occupied }
var i: integer;
    temp: boolean;
begin
  temp:= TRUE;
  for i:= 1 to 9 do
    if board[i]=0 then temp:= FALSE;
  Tie_Game:= temp;
end; { of Function Tie_Game }

Procedure Do_Comp_Wins;
{ does whatever should be done when the computer wins.
  '(click mouse to continue)' message before new game.  }
begin
  Draw_String(104,150,'Computer Wins');
  Correct_Sound;
  get_button;
end; { of Procedure Do_Comp_Wins }

Procedure Do_Tie_Game;
{ does what should be done when it's a tie game.
  '(click mouse to continue)' message before new game.  }
begin
  Draw_String(124,150,'Tie Game');
  get_button;
end; { Of Procedure Do_Tie_Game }

Procedure Do_Human_Wins;
{ human has won }
begin
  Draw_String(128,150,'You Win');
  Happy_Sound;
  get_button;
end; { of procedure Do_human_wins }

Function Three_Moves: boolean;
var i,j: integer;
begin
  j:=0;
  for i:=1 to 9 do
    if board[i]<>0 then j:=j+1;
  Three_moves:= (j = 3);
end; { of Function Three_Moves }

Function Diagonals(var vacant: integer): boolean;
{ returns true if either diagonal contains two player moves with a
  computer move in the center.  vacant returns space to move. }
var temp: boolean;
begin
  temp:= FALSE;
```

THE GAME CUPBOARD

```
    If three_moves then begin
    If ((board[3]=1) AND (board[7]=1) AND (board[5]=2))
      then begin
        temp:= TRUE;
        If board[6]=0 then vacant:= &
          else if board[2]=0 then vacant:= 2;
        end
      else if ((board[1]=1) AND (board[9]=1) AND (board[5]=2))
        then begin
          temp:=TRUE;
          If board[2]=0 then vacant:=2
            else if board[6]=0 then vacant:=6;
        end;
    end;
    Diagonals:= temp;
end; { of Function Diagonals }

Function Special_Sit(var vacant: integer): boolean;
{ returns true if any of the special situations are true.  vacant
  returns the space to move.  }
begin
  Special_Sit:=FALSE;
  If (board[5]=2) AND (Three_Moves) then
    if ((board[2]=1) AND (board[9]=1)) then
      begin
        Special_Sit:=TRUE;
        vacant:=3
      end
    else if ((board[6]=1) AND (board[7]=1)) then
      begin
        Special_Sit:=TRUE;
        vacant:=9
      end
    else if ((board[3]=1) AND (board[4]=1)) then
      begin
        Special_Sit:=TRUE;
        vacant:=1
      end
    else if ((board[1]=1) AND (board[8]=1)) then
      begin
        Special_Sit:=TRUE;
        vacant:=7;
      end
    else if ((board[4]=1) AND (board[9]=1)) then
      begin
        Special_Sit:=TRUE;
        vacant:=1;
      end
    else if ((board[8]=1) AND (board[3]=1)) then
      begin
        Special_Sit:=TRUE;
        vacant:=7;
      end
    else if ((board[8]=1) AND (board[6]=1)) then
      begin
        Special_Sit:=TRUE;
        vacant:=9;
      end;
end; { of Function Special_Sit }

Procedure choice_move;
{ computer can go in a corner or adjacent to a 1. }
var i: integer;
begin
  { free corner ? }
  If ((board[1]=0) and (board[9]=0)) then comp_move[1]
  else if ((board[3]=0) and (board[7]=0)) then comp_move[3]
  else if board[1]=0 then comp_move[1]
  else if board[3]=0 then comp_move[3]
  else if board[7]=0 then comp_move[7]
  else if board[9]=0 then comp_move[9]

  { adjacent space to a 1 ? }
  else begin
    i:=2;
    while(board[i-1]<>1)AND(i<9) do i:=i+1;
    If (board[i]=0) then comp_move[i]
      else begin
        i:=8;
        Repeat
          i:=i+1;
        Until (board[i]=0);
        comp_move[i];
      end;
  end;
end; { of Procedure choice_move }

Procedure Noughts;
var
  vacant,x: integer;
  exit_noughts, comp_turn, harder: boolean;
begin
  exit_noughts:= FALSE; harder:= TRUE;
  Grow_Box(110,159, 76, 36, 0, 0, 320, 200);
Repeat
  Hide_Mouse;
  ScrRest(Nought_Scr);

  Line_Color(black);              { outline harder or easier }
  If harder then begin
    Line(251,199,302,199);
    Line(251,189,302,189);
  end else begin
    Line(16,189,64,189);
    Line(16,199,64,199);
  end;
  Show_Mouse;

  For x:= 1 to 9 do               { clear board }
    board[x]:= 0;
  Get_Move(exit_noughts, harder);
  If NOT exit_noughts then begin
```

THE GAME CUPBOARD

END

```
      If board[5]=0                    ( the center is vacant )
        then comp_move(5)              ( take the center )
        else comp_move(3);             ( take a corner )
      comp_turn:= FALSE;
      Repeat
        If comp_turn then              ( computer makes move )
          begin
            comp_turn:= FALSE;
            If Win_Space(vacant)           ( If computer can win, win )
              then comp_move(vacant)
            else If Block_needed(vacant) ( If comp needs to block, block )
              then comp_move(vacant)
            else If Diagonals(vacant)      ( make a good move )
              then comp_move(vacant)
            else if (Special_Bit(vacant) AND harder)
              then comp_move(vacant)
            else choice_move;
          end
        else begin                     ( human makes move )
          comp_turn:= TRUE;
          Get_Move(exit_noughts, harder);
        end;
      Until Comp_Wins or Tie_Game or exit_noughts or human_wins;
      If Comp_Wins then Do_Comp_Wins
      else if Tie_Game then Do_Tie_Game
      else if human_wins then do_human_wins;
    end;
  Until exit_noughts;

  Hide_mouse;
  ScrRest(Cupboard_Scr);
  Shrink_Box(0,0,320,200,118,159,76,30);
  Show_mouse;
end; ( of Procedure Noughts )

Function Main_Menu: Integer;
( Gets input from main screen.
  Returns: 1=Repeat, 2=Eight ball, 3=Noughts, 4=Exit_Game. )
Const
  Rpt_Top       = 88;    ( These define the shelves of the cupboard )
  Rpt_Bottom    = 118;
  Eight_Top     = 124;
  Eight_Bottom  = 154;
  Nought_Top    = 159;
  Nought_Bottom = 189;
  Left          = 118;
  Right         = 194;
  Exit_Top      = 184;
  Exit_Bottom   = 194;
  Exit_Left     = 241;
  Exit_Right    = 282;

Var
  done: boolean;
  x, y: integer;
begin
  done:= FALSE;
  Repeat
    Main_Event(100);
    If button then begin      ( check to see if item selected )
      x:= Mouse_x;
      y:= Mouse_y;
      If ((x >= Left) AND (x <= Right)) then
        If (y >= Rpt_Top) AND (y <= Rpt_Bottom)) then begin
          done:= TRUE;
          Main_Menu:= 1;
        end else
        If ((y >= Eight_Top) AND (y <= Eight_Bottom)) then begin
          done:= TRUE;
          Main_Menu:= 2;
        end else
        if ((y >= Nought_Top) AND (y <= Nought_Bottom)) then begin
          done:= TRUE;
          Main_menu:= 3;
        end;
      If ((x >= Exit_Left) AND (x <= Exit_Right) AND (y >= Exit_Top)
        AND (y <= Exit_Bottom))
        then begin
          done:=TRUE;
          Main_Menu:= 4;
        end;
    end;
  Until done;
end; ( of Function Main_Menu )

Procedure Main_Loop;
( This is the main loop which calls all of the other routines )
var
  picked: integer;
begin
  Repeat
    picked:= Main_Menu;
    case picked of
      1: Repeat_Game;
      2: Eight_Ball;
      3: Noughts;
      4: Get_Out;
    end;
  Until Exit_Game;
end; ( of procedure Main_Loop )


begin
If Init_Gen>=0 then begin
 Initialize;
 Main_Loop;
 Exit_Gen;
end;
end.
```

LDW Power

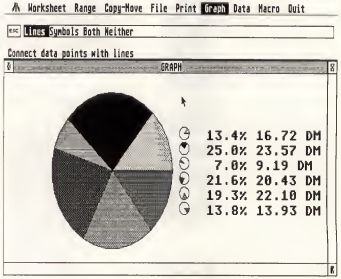
Logical Design Works
780 Montague Expressway
Suite 403
San Jose, CA 95131
(408) 435-1445
$79.95, high and medium resolution

Reviewed by Frank Cohen

The ST is an amazing computer when consider-ing its price and, recently, its software library. The most popular MS-DOS database, *dBase III Plus*, runs on the ST under *dBMan*. And recently, the most popular MS-DOS spreadsheet, *Lotus 1-2-3* (Version 2), runs on the ST under the new *LDW Power* spreadsheet.

While Logical Design Works (LDW) might want to shy away from calling its product a *Lotus 1-2-3* clone (think of how Lotus' lawyers might react), *LDW Power* is an exact replica of that MS-DOS spreadsheet. The two programs are so similar that you can take data and macro programs from the PC to the ST, and vice versa. And the ST version runs more than twice as fast.

This could provide quite an improvement in the ST's standing among the business executives who have snubbed the ST in the past. Imagine a purchasing agent trying to decide between the purchase of a $2,500 basic MS-DOS computer clone, versus the price of a $1,300 ST computer. The savings in the software library are just as dramatic: the latest version of *Lotus 1-2-3* is sold in Computerland stores for $340 with *LDW Power*, holding a list price of only $79.95. The street price of *LDW Power* is even lower.



Previously, ST owners looking for a *Lotus 1-2-3* platfo n had to purchase *VIP Professional* at $249.95. In its time, *VIP Professional* was a good workhorse. ST users could use *VIP*'s many *Lotus*-compatible commands without having to learn a completely new system. But, while *VIP Professional* was a powerful program, it did not support features built into the more recent versions of *Lotus 1-2-3*. *LDW Power* does.

How do you measure a spreadsheet?

Spreadsheets are most easily compared in terms of size. *LDW Power* has a small spreadsheet, 8,192 rows by 256 columns, compared to its MS-DOS look-alike. Don't worry; although some *Lotus* aficionados might think this is a small spreadsheet, it should suffice for just about every ST application.

A possible reason for the size of the spreadsheet is the huge size of the program. *Lotus 1-2-3* has evolved into a powerful piece of software. By modeling *LDW Power* after *Lotus*, the programmers have had to create a program that supports graphics, database functions, a macro-command language and support the mouse/window/menu system of GEM.

LDW Power makes full use of the ST's capabilities. The fast processing speed of the ST gives

this program a great advantage over *Lotus* running on an IBM PC or AT computer. The easiest way to see the speed improvements is to scroll around a fairly complex worksheet. Using a worksheet 20 columns wide and 20 rows tall, *LDW Power* blazes through the columns left-to-right and rolls the worksheet extremely fast, compared to the same simple functions on the MS-DOS machine. When the worksheet is expanded to 100 columns by 100 rows, the scroll speed does not appreciably slow down.

Recalculation speed, also, seems to stay fairly constant throughout several tests. Using the Savage Benchmark test (*BYTE*, June 1987), *LDW Power* processed 1,000 iterations of a complex mathematical formula in 39 seconds. The same test using *Lotus 1-2-3* on an IBM PC finished in 210 seconds.

The measure of an ST application

LDW Power delivers the power, speed and compatibility of its MS-DOS twin. But just how good is it to use on the ST? How good a program "feels" on its host computer can make or break a product. An understanding of *LDW Power*'s other functions is important before making a software purchase.

Comparing *LDW Power* to the other ST spreadsheets does not yield a good understanding of the current state of the spreadsheet market. *LDW Power* is superior in so many ways to its competition: speed, functions, ease-of-use, macros, compatibility and printer support. So the issues of GEM usage, unusual functions and uniqueness are important in a fair comparison.

For example, while *LDW Power* does everything under the sun at extremely fast speeds, it does not work on a 520 ST without a memory upgrade. Once the 345K program is loaded, several memory buffers are established, and the worksheet window appears. There is only enough memory on a 520 system for 85 cells.

In addition, *LDW Power* is disk hungry. During several trial runs, worksheet files containing less than 100 cells occupied more than 5K of disk storage. The Savage Benchmark, which used 1,000 cells, required 120K of storage. *LDW Power* really needs a hard disk drive to be useful to the power user.

The programming design behind *LDW Power* is refreshing. The program is of the everything-fits-in-memory variety. Once loaded, the program diskette may be removed and a data disk inserted.

The minimum hardware setup for LDW Power should be a 1040 ST system with one double-sided disk drive. ST users who frequently switch between programs—word processor to spreadsheet, etc.—should have a hard disk to use *LDW Power* since the program is so large, making it impractical to use one of the program switchers, like *Juggler*.

Once running, the program offers few disk-manipulation commands. A file-delete function is available, but *LDW Power* lacks a file copy, initialize disk and file move functions. *Power* is compatible with the *Universal Item Selector*, a commercially available file-selector alternative that provides all these missing functions.

The look and feel

A program of this size should feel comfortable to use. Since each ST owner has a different style of usage, the program should conform to the user. But, most of the time, this happens in reverse. Fortunately, *LDW Power* has several subtle features that make it very easy to use.

Upon exiting *LDW Power*, a special file is stored to keep the user settings established during the work session. For example, once the default file directory of a hard disk has been established, *LDW Power* saves a special file, LDWCNF, to be read the next time the program is used. This is one of those nice extras that makes a piece of software more comfortable to use, but one that is often overlooked.

For *LDW Power* users that work with one spreadsheet most of the time, any worksheet titled AUTO.LDW will be automatically loaded when *LDW Power* is first started.

The mouse can be used to select a range of cells for an operation, incorporating the classic "click and drag" to determine the range. If the range extends outside of the current window size of the worksheet, the window automatically scrolls in the needed direction to show the extended drag region. This function works in reverse, as well.

The layout of the screen is comfortable: the usual bag of GEM tricks, with a few new controls to add a little spice. The menu bar is designed to emulate *Lotus 1-2-3* Circular menus. The Quit entry is a drop-down menu title on the right of the screen. Since most ST users expect Quit to be the last entry in the File drop-down menu, it might take a little getting used to, but the menu layout is skillfully designed.

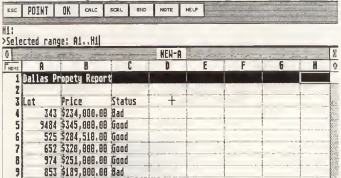
Lotus 1-2-3 does not have the GEM system to draw from, so each command is keyboard driven, although, just recently, versions of *Lotus* have appeared that lightly support a mouse and windowing system. The keyboard menu commands all begin with the "/" character. Pressing the "/" key in *LDW power* activates the menu bar. Each menu entry uses a separate keyboard equivalent, usually the first letter of the function. For example, pressing "/Q" quits *LDW Power*. Pressing "/P"

Lotus 1-2-3 does not have the GEM system to draw from, so each command is keyboard driven, although, just recently, versions of *Lotus* have appeared that lightly support a mouse and windowing system.

prints the worksheet.

LDW Power is uniquely good at supporting the keyboard and mouse. After pressing the "/" key, the mouse can also be used to click on the desired menu function. In fact, the mouse and keyboard can be used interchangeably at any time. The left mouse button selects a function, while the right mouse button is the equivalent of pressing the Return key. First-time users will spend an hour getting used to the new system.

Directly beneath the menu bar is a row of eight command buttons. *Lotus* users comfortable with using the keyboard will find the command buttons an easy method of accessing all of *LDW Power*'s



commands. ST users who rely on the mouse will most likely not use the command buttons, as their drop-down menu equivalents are always available in the usual GEM menu bar.

One of the command buttons, SCRL, puts *LDW Power* into scroll mode. When active, the worksheet can be scrolled up/down and left/right using the arrow keys. This function isn't needed by mouse users, as the worksheet window has the usual GEM horizontal and vertical slider bars.

LDW Power works with desk accessories, so it is also possible to use a calculator or other supporting program. The *LDW Power* windows can be dragged and sized; however, the desktop area

Printing a report with LDW Power is not a simple function. Since there are so many options, a good read through the manual is needed before the novice user will feel comfortable.

of GEM has been reduced to prevent *LDW Power* windows from being dragged over the command buttons. This feature reduces the amount of screen display area to 14 rows of worksheet data. It would have been better to allow the command button area to be covered, if so desired. To help make up for this, Monochrome ST computers can use a condensed type style that displays 29 rows of worksheet data; however, many users will find the condensed screen font straining on the eyes.

Last year proved to be the year of the accessory. When testing *LDW Power*, *Turbo ST* greatly increased the overall display speed. *Turbo ST* is a software blitter emulator that improves the screen drawing speed of GEM. In addition, *G+Plus* improved *LDW Power's* speed when printing drawings to a GDOS-compatible printer. *G+Plus* is a replacement for Atari's GDOS operating system software. Both products are good investments when using *LDW Power*; however, they both rob the *LDW Power* worksheet of valuable memory space.

The look of a worksheet can be enhanced by using some of GEM's text effects. Bold and underline can be used to highlight important areas of a spreadsheet.

The one noteworthy feature missing from *LDW Power* is the ability to correct mistakes made using the program's functions. The ST keyboard has an Undo key, but few ST program support its use. In theory, every GEM program should be able to reverse the effects of the last command. For some unknown reason, *LDW Power* skipped this important feature.

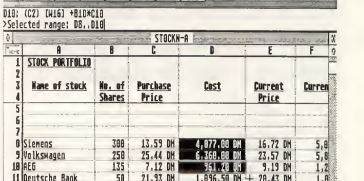
Macros and functions

At first glance, a spreadsheet appears to be a monomaniacal program. Its sole purpose is to crunch numbers using a metaphor established by accountants: the worksheet. Macros and functions make a spreadsheet much more than a simple computerized version of an accountant's pencil-pushing task. Functions allow data to be tested, and the results of the test can alter the contents of the worksheet or additional spreadsheet functions. Macros make it possible for the spreadsheet to learn about and analyze the information that is entered into the worksheet.

Suppose the spreadsheet is used to enter real-estate information into an ST computer. Since computer unsophisticates are usually used for data entry, the resulting precision of the work can leave much to be desired. Using a spreadsheet macro, the data entered can be tested against predetermined values. If the data entered is bad, the data-entry person can be alerted and asked to enter the information again.

Suppose Column A of a spreadsheet is used to enter a series of dollar amounts for houses located in a suburban area of Dallas. The real-estate agency using the spreadsheet has determined that houses costing less than 20% of the average of the other homes entered is a bad risk. After entering the house values, the agency manager wants to see a list of the good and bad homes. To makes this work, Column A is established to receive the name of the house, Column B holds the price, and Column C displays a GOOD or BAD indicator.

Column C displays the word GOOD or BAD if the average of the first ten lines of Column B is 20% higher than the last entered home value. Column C's formula contains



the following expression:

```
@IF (@AVG(B2..B10) > B10*1.20, "BAD", "GOOD")
```

The IF function tests the first expression, AVG, and yields either a GOOD text value if the average is greater than the house or a BAD text value if the average is less than or equal. LDW Power has a large library of functions, making it easy to create complex, intelligent worksheets.

The IF function used in this example works on one cell at a time. Macros automate functions by creating a template of commands. The macro function works like a tape recorder. When recording a macro, the mouse or keyboard is used to move through the worksheet and call functions. When a macro is played back, the user sees the worksheet cursor move through the functions that were previously recorded. A macro used in the home-pricing example moves the cursor from one row to another, allowing the user to enter a home value. The last macro command might produce the printed report requested by the office manager.

Reports are complex

Printing a report with *LDW Power* is not a simple function. Since there are so many options, a good read through the manual is needed before the novice user will feel comfortable.

Like *Lotus 1-2-3*, the report-writing capabilities are strong in comparison to other ST spreadsheets. Options include customized headers and footers, page numbering with date and time tags, adjustable margins and borders, user-definable page length and alignment, etc. All selected print format commands are saved with the worksheet.

The program supports GDOS, assuring what-you-see-is-what-you-get output when using a GDOS-compatible printer. The list of GDOS printer drivers is short, so be sure to check with LDW for printer compatibility *before* buying. *LDW Power* uses a straightforward ASCII printer driver to print the contents of a worksheet.

The user can choose output to be sent to the parallel or serial connectors of the ST. Line feeds can be included with carriage return characters, making it compatible with IBM printers. Output can also be sent to a disk file, for later modification using a word processor.

A special utility is included that allows *LDW Power* to print worksheets and other ASCII files at a 90 degree angle. The equivalent program in the MS-DOS world is called *Sideways*. The LDW utility allows spreadsheets wider than the standard 8½×11-inch piece of paper. This is a must for business users.

information into *LDW Power* is in the disk drives. The information has to at some point be put onto a 3½-inch floppy drive. Most new IBM computers are being shipped with 3½-inch and 5¼-inch floppy drives, but if your equipment doesn't have this capability, try contacting a local IBM dealer. They usually will copy Lotus files onto a 3½-inch disk at no charge.

LDW Power supports mass import of text and numbers from non-*LDW Power* files, the data being loaded into the worksheet as text labels. The database functions can later be used to separate the imported data into rows and columns. A similar system can be used with raw numeric data.

to support password protection, a feature necessary when selling spreadsheet software to government institutions or the Fortune 500.

A word on support

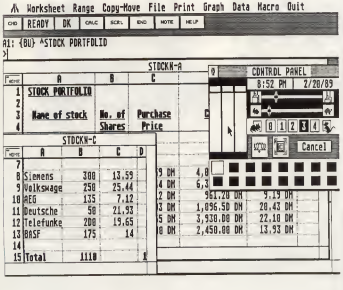
The word is "good." Overall, LDW has supported the ST community with technical support, dealer support, software upgrades and decent documentation, and the company seems to draw enough revenue from Europe to make a good run of its U.S. operations. LDW has been visible at the latest trade shows and appears have a good future ahead.

The *LDW Power* manual is adequate to show you the ropes; however, overall, the documentation is only fair. The text is informative as a reference, but more pictures and illustrations are needed. Luckily, since *Lotus 1-2-3* has such an established name, most book stores carry a complete section of tutorials, primers and other reference books that, due to its similarity to *Lotus*, also apply to *LDW Power*.

LDW's technical support is first rate. You can reach them at their San Jose, California offices weekdays during business hours.

Conclusion

If you're looking for a spreadsheet program that'll give you not only the power of *Lotus 1-2-3*, but also compatibility with that popular program, then you need look no further than *LDW Power*. ■



Communicating with the rest of the world

One of *LDW Power*'s biggest features is the ability to directly load *Lotus* worksheets and macros. MS-DOS users that are equipped with 3½-inch floppy disk drives are in luck. Since the ST floppy diskettes are directly compatible with IBM's disk format, to load a Lotus .WK1 file into LDW Power is as simple as inserting the diskette into your ST drive, and selecting Retrieve from the File drop-down menu. *LDW Power* takes care of the translation and promptly displays the Lotus file.

The key to getting the *Lotus* information into *LDW Power* is in the disk drives. The information has to at some point be put onto a 3½-inch floppy drive. Most new IBM computers are being shipped with 3½-inch and 5¼-inch floppy drives, but if your equipment doesn't have this capability, try contacting a local IBM dealer. They usually will copy Lotus files onto a 3½-inch disk at no charge.

LDW Power supports mass import of text and numbers from non-*LDW Power* files, the data being loaded into the worksheet as text labels. The database functions can later be used to separate the imported data into rows and columns. A similar system can be used with raw numeric data.

Hopefully, *LDW* will eventually release a utility program that imports and exports data from several other formats: delimited, SILK, DIF, etc.

Of course, worksheet information may be stored to a disk file. In addition, *LDW Power* has several sophisticated commands that allow portions of a worksheet to be merged with other worksheets. Complex worksheets can use data contained in external worksheets, a handy function when the maximum number of cells is reached.

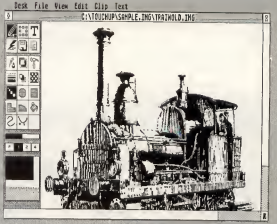
Worksheets can be secured with the use of a password, up to 15 characters long. *A-Calc Prime* is the only other ST spreadsheet

Frank Cohen *regularly contributes to* ST-LOG *and ANALDG. His experience with Atari computers stretches back to his first commercial product* Clowns & Balloons. *He may be contacted on CompuServe (76004,1573) and GEnie (FCDHEN), or directly at P.O. Box 14628, Long Beach, CA 90803-1208.*

Touch-Up

Migraph, Inc.
200 S. 333rd St.
Suite 220
Federal Way, WA 98003
(206) 838-4677
$179.95

Reviewed by David Plotkin



Paint packages such as Electronic Arts's *DEGAS Elite* and Atari's *Neochrome* have done well and have been used to create a remarkable variety of art. These packages have one major limitation however: They can create only graphics with the same resolution as the screen. *DEGAS Elite* thus has a limit of 640×400 in monochrome mode, or about 80 dpi. Most printers are capable of higher resolution however—from about 100 dpi for inexpensive 9-pin dot-matrix printers, up to 300 dpi for laser printers and the H-P Deskjet.

There is a popular file format, an "image" file, which supports such high resolutions. These files generally have an extender of .IMG and can be in virtually any resolution. Most desktop publishing packages, as well as Migraph's *Easy Draw*, can import and use .IMG files. However, except for a few scanners and some commercial clip art, there has not been any source of .IMG files, and no way to edit the few that were available. That has all changed with the introduction of *Touch-Up*, the high resolution graphics editor from Migraph.

Touch-Up's main screen consists of a working window, a tool panel on the left side of the screen and a complete set of drop-down menus. The working area shows the graphic you are working on. Since the resolution of the graphic can be far higher than what can be shown on the screen, *Touch-Up* compensates by showing the graphic larger than full size, and the graphic can cover many screens, provided you have the memory available. The working area can be scrolled just like any window, using the scroll bars to reach the part of your picture you want to work on. You can also scroll by clicking and dragging the "locator," a black box that represents what you can see on the screen within a white box that represents the whole graphic.

The tool panel is normally fixed to the left side of the screen, but it can be hidden to provide a larger working surface or detached into a window of its own, so that it can be relocated on the screen. It is often advantageous to hide the tool panel, since many of its functions are duplicated in the drop-down menus. As is normal for such programs, all drawing is done with the mouse. A variety of drawing tools are available, including box, circle, polyline, arcs, freehand sketch and splines. Upon selecting the box or circle, a box or circle appears instead of the mouse cursor, in whatever was the last size you used. You can resize the brush by moving the mouse with the left button pressed. The brush moves with the mouse until you press the right button (as in *Easy Draw*) to paste the brush down. When the brush is pasted down, it is then drewn with the current line style and fill.

Splines are an interesting and powerful tool. A spline is a curve that attempts to follow a series of control points, which you lay down by clicking the mouse's left but-

ton. Before transforming the spline into a brush, you can edit the control points (remove or move them). Once you have created a brush with the spline, you can still edit points, using the keys and mouse. Even the "goodness of fit" (how closely the spline tries to pass through your specified control points) is user-adjustable.

Special-effect and user-definable settings abound in *Touch-Up*. The line width and end type (arrows or plain), as well as whether edges are drawn for filled shapes, can be set. Eight different shadow offset directions can be chosen, and the offset for the shadow can be selected as well. Four different "writing modes" can be used to create some special effects: replace, XOR, transparent and reverse transparent. All 36 GEM patterns can be used for fills; however, there is no user-defined pattern. Four zoom magnifications are available ranging from full (you can see the whole page represented on the screen) up to four times normal.

Touch-Up's "lightning" mode gives you access to only the currently visible screen (no scrolling to other parts of the diagram). However, any changes you make in lightning mode are automatically transferred to the main diagram when you exit.

There are quite a few advantages to working in the lightning mode. First of all, there is an Undo feature, which is missing from the regular mode (and sorely missed). You can Undo the last action or all changes made since entering lightning mode. Then there are the additional drawing tools. An airbrush is available, which can be square or circular, in one of three sizes. The maximum saturation of the brush can be set, as can the fill speed. You can also decide to use the current pattern for the fill.

Another new tool is the "fat bits" mode, which provides a small portion of the screen in extreme magnification, so that you can work on individual dots. Lasso is another useful tool. With it, you can lasso just about any portion of the picture and start using it as your current brush.

The resizable clip rectangle provides still more options. Once it has been specified, it allows you to move or copy whatever it surrounds to another place in the diagram. Its contents can also be saved to and loaded from disk, providing a rudimentary clipboard capability. Besides using .IMG files, the clip area can be saved as a *DEGAS*, *MacPaint*, GIF, PCX, TIF or IFF-ILBM files. You may load most of these files into the clip area as well.

A utility provided with *Touch-Up* can convert *PrintMaster* files to .IMG files and back again. Thus, *Touch-Up* can edit and save *PrintMaster* files—a significant plus. The clip area can also load .GEM files from programs such as *Easy Draw*; unfortunately, the text is not loaded, and therefore is lost. This is too bad because many .GEM files include text. This is rather a serious limitation.

The contents of the clip rectangle can be mirrored, flipped, slanted and rotated. Three other special effects are also available—Cleanup (removes stray dots), Outline (removes fill patterns, leaving just the shape outline) and Mask (creates special effects using fill patterns, generally to lighten them). Finally, you can use the image viewer to look at any .IMG file on the disk without loading it, and even set the clip rectangle to be the proper size so that you can then load it without distortion.

Touch-Up does text. It comes complete with ten special fonts, all of which can be set to be any size from too small to read up to 999 dots (not points) high. Various attributes, such as underlined, bold, slanted, italic, fat, outlined and filled can be specified. Filled is especially powerful, since the letters will be filled with the current fill pattern.

Using text is somewhat awkward however. You must specify the size and attributes, then click on an icon that lets you type up to 35 characters into a dialog box. Once you have done this, you move the cursor onto the drawing area and click the left button, which will bring up a box that is roughly (very roughly) the size of the text string. Clicking the right button then pastes the text down and draws the letters. It's not a word processor, but it does work.

Although *Touch-Up* produces only black-and-white pictures, it can import color pictures, converting them into black and white. There are a variety of algorithms you can use for this conversion, ranging from simple (quick and dirty) to 4×4 table mapping, where each color is mapped to a different 4×4 grid pattern. The latter takes awhile, and you can end up with a *big* image file (takes up lots of memory and disk space). The results are remarkable, however, especially at 300 dpi on a laser printer.

Touch-Up is flexible when setting up an image. A dialog box allows setting the size of the image (in pixels, centimeters or inches), the size and location of the clip rectangle (although it's easier just to drag it where you need it) and the resolution in either dpi or pixel size. Unfortunately, the manual is incorrect on how to set the pixel size. You cannot simply type in the numbers you want and click on OK. You must first click on the DPI button (even if it is highlighted already) or your adjustments will be ignored.

Speaking of the manual, in general, it's well written, although slightly disorganized. There are sections (as noted in the previous paragraph) where explanations are wrong, as if the person writing the documentation didn't get a chance to test it against the final product. There are also inconsistencies in the product itself. For example, when drawing boxes in regular mode, you get rounded corners by holding the Shift key. However, when drawing boxes in the lightning mode, you must press the Alternate-R toggle to get the rounded corners. Why should these be different? There are also instances where two similar actions are not selected the same way. Such anomalies are not serious, but are annoying, since you must drag out the (indexless) manual to try to figure out how to perform the task.

Touch-Up's disks are not protected, and it runs just fine from a hard drive. However, Migraph has been hurt by piracy and so has taken the rather drastic step of providing a hardware "key" that must be plugged into the printer port. While some may be irritated at having to pay for such a thing (keys are not cheap, that's why many manufacturers do not use them), my reaction is that if this is what it takes for Migraph to continue to provide the excellent software (and this includes *Touch-Up*) they have in the past, it is a small price to pay.

Touch-Up is a GDOS application and comes with several utility programs that are useful. First, there's OUTPRINT, which can print any .IMG file. It is the same program provided with *Easy Draw*. Since a small .GEM file must be present for OUTPRINT to work, you cannot print an .IMG file until it has been loaded into *Touch-Up* and saved again. As mentioned earlier, a program for converting *PrintMaster* libraries to .IMG files is also provided, which makes it easy to use *PrintMaster* icons in your desktop-publishing files. Lastly, a utility that speeds up your mouse is included. *Touch-Up* is completely compatible with *G+Plus* from CodeHead software, but is not compatible with *Turbo ST* from Softrek (*Turbo ST* fouls up the flip and rotate functions of the clip area—at least).

Touch-Up is a powerful drawing package that can handle .IMG files, as well as many popular graphics formats. Because of this power, you will need to spend some time learning to use the program, but the time spent will be well worth the effort. This program is highly recommended.■

***David Plotkin** has been pounding the keys on Atari computers for almost ten years, and in that time he has written many memorable programs and articles. He has an MS in chemical engineering and is a data analyst for Chevron Corporation.*

Prospero Pascal

Prospero Software
100 Commercial Street
Suite 306
Portland, Maine 04101
(207) 874-0382
$149.00

Reviewed by David Plotkin

Prospero Pascal is a full-featured Pascal that includes everything you need to begin writing stand-alone programs for your Atari ST.

Pascal is the language of choice for many programmers of the Atari ST, as well as many other computers. Pascal is a structured language that makes it easier to follow program logic and avoid the pitfalls of "spaghetti coding." The results of a Pascal program are compiled to machine language, and can thus run quickly, as well as be marketed commercially or handed out to people who don't own a copy of the language.

Prospero Pascal is a full-featured Pascal that includes everything you need to begin writing stand-alone programs for your Atari ST. In addition to the language and associated libraries, Pro Pascal includes a symbolic debugger, cross-reference variable generator and "library manager" to help programmers who wish to maintain their own libraries.

Working on the workbench

Pro Pascal is operated from a shell known as the "workbench." Fully menu-driven, the workbench gives you access to a full-function editor with many advanced capabilities, the compiler and linker. You may also run a program straight from the workbench, so you can test the results of your programs without exiting Pro Pascal. Quite a variety of configuration options can be specified and saved to disk for future sessions.

The editor can load a .PAS file (or any other file, for that matter) and runs in a window with scroll bars, arrows and the usual GEM controls. The full features of the editor make it a straightforward task to edit your file. You can specify blocks of text using the mouse (full click-and-drag), and once you have done that, you can cut, copy, paste, delete or write the block to disk. You can also load a block from disk, giving the capability for merging different program fragments. The editor supports find and replace (both forward and backward), auto-indenting loops and decision statements, and can instantly jump to any line number. Even the common *WordStar* com-

mand keys for moving the cursor are supported. Finally, the function keys can be defined to produce just about any string you desire, giving you full macro capabilities.

The compiler converts the Pascal source code to relocatable machine language. It is fast, yielding compilation times comparable to OSS/ICD's Personal Pascal compiler. One of the things that makes the compiler so flexible is the ability to specify the drive and path from which include files, work files and libraries will be read from or written to. The compiler has a separate option to do a syntax check on the source code that is handy and much faster than attempting to do a complete compile, only to find many syntax errors. Many compiler options allow you to customize what you want the compiler to do. Checking (arrays, assignments and pointers) is recommended during program development, these can be turned off for the final compile to produce faster code. Double precision can be used for real numbers, and you can ask for shorter (but slower) code generation.

The linker links the machine language output from the compiler with the files necessary for the program to run. The workbench offers the option of linking with files necessary for running under GEMDOS and GEM (VDI and AES), as well as any other files you desire. The linker can be used with a control file that lists the names of all files that are to be linked, which provides quite a bit of flexibility.

Also included with the workbench is PROBE, the symbolic debugger. This advanced program tests a program and can be invaluable in finding errors. The output of PROBE can be routed to a file, the printer or an alternate screen. A cross-reference generator that compiles a list of all variables (called "source identifiers" in the Pro Pascal documentation) is available. It can even list the variables in any included files and show what line these variables are used on.

The language

Pro Pascal is a complete Pascal, meeting not only the ANSI standards, but also containing many of the "normal" extensions users have come to expect. Full variable typing, including integer, real, char, Boolean, enumerated, subrange, array, record, set, pointer and files are all supported. The files can be either text or nontext, and random-access files are also available. Procedures and functions, with both variable and value parameters are provided, as is CASE, REPEAT, WHILE, FOR, WITH and IF. The standard set of math functions, commands to input and output text, string handling, Boolean (AND, OR, XOR), set operations and bit/byte manipulation are all built into Pro Pascal. Even in-line assembly code is supported.

As with any other version of Pascal, all variables have to be declared, but there is no limit on the number of times the CONST or VAR declarative headers can be used. Segment compiling is supported; that is, procedures and functions can be grouped into a segment or module and compiled separately. By then including the compiled code in the final product (using the EXTERNAL directive), you are freed from having to recompile the (assumed to be) error-free sections of your program over and over, thus saving time and increasing the ease of debugging because the program in the editor will be shorter, since it need not include the segment-compiled portions.

Full array support includes the Pascal extension to denote an array as either "ARRAY[t1] of ARRAY[t2] of t" or "ARRAY[t1, t2] of t," making it conform to the more "normal" notation of arrays. Record types include variant tag fields. String manipulation functions include CONCAT, COPY, INSERT, LENGTH, DELETE, STR and POS, making string handling a powerful part of this language. File-handling command extensions include assigning a file to a device (handy for printing), variable buffer sizing, RAMdisk support, appending to a file, updating (both read and write access) and full random-access text and nontext files.

Pro Pascal also contains commands supporting GEMDOS's ability to allow one executing program to invoke another, complete with message passing and return codes. Although these advanced concepts will not be of use to the majority of programmers, they permit the developer to include powerful capabilities in their Pro Pascal programs. Finally, Pro Pascal has extensions for hyperbolic trig functions, Peek, Poke, address of variables, a prompt function, date and time.

A real GEM

Pro Pascal includes a full range of support for both VDI and AES, in the form of external procedures that match, almost exactly, the format of VDI and AES calls in C. This similarity makes it possible for someone who is just learning how the GEM functions work to make use of the large amount of information available about the C GEM calls. In your Pascal program, you must include the file that declares the special Pascal parameters and arrays, then link the file that contains the external procedures themselves. Virtually every VDI and AES function is included, including all graphics, menus, dialog boxes, windows and events. Each function is accompanied by a complete explanation of how it works, example programs and a declaration of the external procedure or function.

Manual labor

The three manuals that accompany Pro Pascal are high-quality and spiral-bound to allow them to lie flat. They are not designed to teach programming in Pascal, but are excellent reference guides. The first manual is on the language and the workbench (editor, compiler, linker) and includes a full description of the extensions to the language. The two additional manuals, describing the VDI and AES extensions, are complete, and little additional information on GEM should be needed beyond what is supplied.

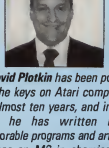
Comparisons

Inevitably, comparisons will be drawn between Pro Pascal and the giant of the ST Pascal world, Personal Pascal from DSS/ICD. Both of these languages are excellent implementations of Pascal, but they are quite different, especially in how they handle GEM. Pro Pascal uses the straight "generic" VDI and AES bindings, while Personal Pascal has its own functions that combine some of the more tedious GEM bindings into more powerful (and easier to use) commands. Personal Pascal allows you to build your own Dialog boxes and menus right in the program. It also automates much of the handling of windows and gives excellent descriptions and sample programs showing how to use these custom commands.

One thing that bears considering is that Pro Pascal is being updated and actively supported by Prospero, while Personal Pascal has languished, with no strong marketing force to keep it viable. Further, although many of the GEM aspects of Personal Pascal are easier to use, they are different from the rest of the ST world, while Pro Pascal has pretty much adhered to the standard.

Conclusion

Prospero Pascal is a complete, high-quality language with superb manuals. The wealth of commands, VDI and AES bindings, a quality editor and the many options and extensions make it an excellent value, and it is highly recommended to the budding (or experienced) Pascal programmer. ■



***David Plotkin** has been pounding the keys on Atari computers for almost ten years, and in that time he has written many memorable programs and articles. He has an MS in chemical engineering and is a data analyst for Chevron Corporation.*

The ST GAM REVIEW

Universal Military Simulator Scenario Disks

Disk 1: The American Civil War
Disk 2: Vietnam

Rainbird Software
P.O. Box 2227
Menlo Park, CA 94026
(415) 322-0412

Reviewed by Frank Eva

The *Universal Military Simulator* (*UMS*) is a software package that will not blow you away graphically. However, it is an efficient tool for the serious war gamer or history buff. For those who are unfamiliar with this program, perhaps a few brief words about the system are in order.

First of all, *UMS* depicts historical conflicts in the medium and high resolution modes of all Atari STs, with at least 512K of RAM. The high resolution mode is, of course, possible only for those with monochrome monitors. The medium resolution mode limits the program to a total of four colors, the defaults being black, white, green and orange. The defaults for high resolution are black and white, the normal output of Atari monochrome monitors. Because of the color limitations, the program relies heavily on grid maps, which are basically black-on-white line drawings that represent contours in the landscape only marginally.

Armies are depicted by unit, with the commanding officer's name emblazoned on the icon. An arrow points to the position of the unit on the map. Additionally, a smaller icon represents the type of unit at that location: i.e., light infantry, heavy artillery, etc.

While limited graphically, *UMS* has the unique ability to depict the setting of each battle from any direction of view selected by the user, as well as the ability to utilize two levels of zoom. However, zooms again do not accomplish anything graphically, aside from the fact that the enlarged areas permit the user to see more clearly what is going on. A zoom merely enlarges the scale of the original map and does not display anything beyond the field of the zoom. It is obvious that *UMS* was not programmed in the style of Chris Crawford, the father of home computer war games.

However, the 3-D grid maps do give the war gamer a feel, at least, for the lay of the land. This is a feature that was lacking in most of the early war games. Additionally, *UMS* provides the user with the ability to create entirely

ESHELF

new scenarios and customized maps. This ability has spawned the next generation of scenario disks for the *UMS* owner.

Scenario Disk #1 includes three major battles from the American Civil War. While the original disk included the battle of Gettysburg, this new disk provides the battles of Shiloh, Chattanooga and Antietam. The battle of Shiloh presents the surprise attack of confederate General Albert Sydney Johnston's 45,000-man army of the Mississippi against the unprepared forces of Ulysses S. Grant. The battle of Chattanooga depicts the besieging confederate army holding the towering high ground, while the union army, with an ever dwindling store of supplies, held the sleepy Tennessee town of Chattanooga.

The battle of Antietam, September 17, 1863, is referred to as the bloodiest day in the history of American arms. Robert E. Lee has gambled the future of the Confederacy on an invasion of the north. The secret battle plans fall into the hands of General George B. McClellan, but the issue would still have to be decided by over 150,000 American troops!

Scenario Disk #2 includes three battles from the Vietnam war: Hill 823, Hill 875 and Ngoh Kam Leat. Hill 823 depicts the 4th division's mission to clear the region southwest of Dak To, since intelligence reported that the North Vietnamese were gearing up for another big thrust there. Hill 875 was crucial to the battle for Dak To. American forces switched from attack to defense, as the NVA crashed down upon them.

On the slopes of Ngoh Kam Leat, American forces under Captain Thomas Baird stumbled into a hornet's nest of enemy opposition. Despite constant sniper fire, ground attacks and significant casualties, the American troops prevailed.

The procedure for using scenario disks is simple. The user loads *UMS* as usual, supplying a password from the original scenario handbook to verify ownership, and then clicks on the "run simulation" button. This brings up a selection screen that displays the original scenarios and a button in the lower right-hand corner to display scenarios contained on another disk. Clicking on this button brings up a standard file selector. Selecting a file will load the simulation from the supplementary disk. From here on, all *UMS* functions are exactly the same as the original.

While limited graphically, *UMS* has the unique ability to depict the setting of each battle from any direction of view selected by the user, as well as the ability to utilize two levels of zoom.

By the way, the scenario disks are not copy protected. So, they may be copied for archival purposes or loaded onto a hard drive.

All scenario disk packages are accompanied by a detailed scenario handbook. *The American Civil War* is 80 pages; *The Vietnam War* is 30 pages. It is obvious to this reviewer that many hundreds of hours went into the preparation of these supplementary disks and their accompanying documentation. Only the most dedicated history buff/war gamer would be willing to devote so much time to such an occupation. Now, Rainbird has taken all the work out of the process for the majority of *UMS* owners, and done it at a very reasonable price! Scenario disks have certainly improved the longevity of a fine tool!

The bottom line: A good value for owners of *UMS*.

Menace

Psygnosis Ltd.
First Floor
Port of Liverpool Building
Liverpool L3 1BY
United Kingdom
$29.95, color only

Reviewed by Frank Eva

In the finest tradition of horizontally scrolling arcade games such as *Defender*, *Gradius* and *Life Force*, *Menace* explodes on the 68000 microcomputer scene. Using the advanced graphics and speed available on the Atari ST family of home computers, Psygnosis has brought arcade enthusiasts a wonderful diversion from the action/strategy games they have become famous for. At last, an intense, albeit mindless, shoot 'em up that truly utilizes the 68000 microchip to its fullest!

Menace's documentation informs the player that the inhabitants of the planet Draconia have become the most feared and ruthless plunderers of known space. Ousted from their home planets, they have formed an alliance of terror and have taken the planet Draconia as their home base. You must put an end to their reign of terror.

While a large-scale attack might prove too costly, the free worlds are always willing to sacrifice a single ship and its pilot—*you!* Also, it is assumed that a single attacker might be able to approach almost undetected. For this reason, a space slug has been captured. Your ship will hide in the maw of the slug. It will then be controlled remotely, to the planet Draconia. At the appropriate time, a signal will be sent that will cause the slug's mouth to open, at which time you will be free to engage the enemy.

Draconia consists of six zones; Sea of Karnaugh, Vanguard Warzone, Carnage Rift, Tropics of Mace, Ruins of Kruger and Plateaus of Draconia. Destruction of all six will leave the planet totally vulnerable to attack.

As mentioned, *Menace* provides six levels of progressive difficulty. In addition, two difficulty settings are possible: novice and expert. In the novice level, only collisions with aliens and guardians reduce your shields. In the expert level, collisions with scenery will also reduce your shields. What this really means is that there are twelve levels of progressive difficulty. There are 60 different animated aliens to conquer along the way, and an all-powerful "guardian" to combat at the end of each level. All of these features considered, *Menace* should give you many hours of interesting play.

Final Assault

Epyx
600 Balveston Drive
Redwood City, CA 94063
$29.95, color

Reviewed by Steve Panak

I'm really not sure how to start in on this one, so I'll come right to the point. *Final Assault* is a simulation that attempts to recreate the thrills, spills and chills of mountain climbing. Technically, it succeeds on nearly every level. However, subjectively, I did not care much for it, feeling, as I do toward most simulations of this type (i.e., a fishing simulation I played about a year ago): that such an activity does not lend itself to computerization. Put simply, although technically brilliant, something is lost in the translation.

This said, those still interested will be pleased to hear that *Final Assault* does a great job. As would be done in preparation for a real climb, your first task is to assemble in your backpack items you feel will be of use. The items selected will depend on the time of day and length of your climb, as well as the season and characteristics of the summit. The weight of each item is taken into account, so don't overload yourself. For instance, a long climb would require food, perhaps a tent for shelter. Icy climbs require crampons (boot spikes) while rock climbing might be easier with soft shoes.

There are a number of trails available on three mountain ranges, enough to keep most climbers busy until they are quite tired of the program. Although the manual included a short section on the Alps, one of the most famous mountain ranges in the world, I was disappointed to discover that the actual mountains you climb seem to have no basis in reality. However, each is rated as to difficulty and estimated time for completion. After choosing your trail and loading up with gear, you head for the hills.

Play is divided into two main classifications: walking and climb-

Menace uses ultra-smooth parallax scrolling to paint its intriguing screens. Parallax scrolling refers to the slight difference in scrolling speed between the foreground end background. Your eyes perceive this difference and translate it into greater depth, the impression of a third dimension. Disney Studios pioneered this technique, which then gave its flat cartoon animation some feeling of depth.

Menace features a continuous sound track that can be disabled when it starts to grate on the nerves. Sampled effects and speech are also included.

Both mouse and joystick controls are allowed. Each are good, and neither present any real difficulties for the gamer. The high scores can be saved to Disk B of the two disk set, if the write protect tab is pleced to "write enable." There is a nice restart feature that arcade purists will really appreciate. As with arcade games such as *Lock-On*, the player is allowed to restart at the current level in which his last ship was lost. Any special weapons collected along the way are lost, however, if this option is selected.

Another nice feature is the aforementioned ability to collect superior weapons, for use on your own craft. The manual justifies this by saying that you have on board the latest in matter converter weapons. These weapons allow you to change the molecular structure of space debris by continually bombarding it with high-energy shots. By picking up the debris when it is in usable form, you may build and replenish extra weapons. The computer will generate images for you, showing you when to pick up the debris for specific weapons.

Menace

In reality, here is what happens: Destroying an entire wave of attacking Draconians will cause an icon to appear on the screen. Firing at the icon five times will cause it to change into another icon depicting a different weapon. You may collect the weapon depicted simply by flying over the icon.

Icons are provided for additional bonus points, cannons (short-range weapons), lasers (long-range weapons, especially useful against guardians) and outriders. The outriders are probably the most interesting and useful. They are externally mounted droids that can fire at adjustable angles, as opposed to the lasers and cannons that can be fired only horizontally. Finally, icons representing speed boosters, force fields and full shield recovery are available.

While the program disks are copy protected, the *Menace*'s reasonable price makes up for this inconvenience. The animation of the on-screen aliens and attackers is really worth the price of admission. Menace is the kind of game that makes the ST look so much more impressive than Nintendo and Sega home entertainment systems. Not that I am advocating the ST as a game machine, but for those who like games, while really needing more than a static game machine, it's still nice to know that the ST plays the best games available today. Menace rates a solid A!

The bottom line: Buy it.

ing. Climbing is further divided into scaling ice and rock, each of which requires different techniques. The use of a rope might also be necessary on some of the more difficult ridges. On a typical trail, it might be necessary to first hike to the base of the mountain—testing the ground with your ice pick, jumping over crevices as they appear—until you reech an ice cliff. You'll then select the proper gear and start your climb. This scaling works much the way the old *Crazy Climber* arcade game worked, because you use the joystick or keyboard to plece and move your feet and hands. And if you're not careful, don't fret: A beginner gets up to three lives. An expert should know better. Rounding out the program is a save-game feature that allows you to rest in the middle of lengthy climbs, and a log to allow you to record your achievements.

The manual is surprisingly complete and concise; the program remarkably easy to learn. While I usually find that the translation of simple human motions, such as walking or climbing, on the computer requires a frustrating and awkward set of unnatural commands that make learning and playing the game a chore, *Final Assault* escapes this trap. While it is difficult to learn, I was amazed to find that once the motions were mastered, they became almost instinctive, with the result that you improved in skill until you were confident on even the toughest summits. And when combined with the informative manual, the package teaches the neophyte just about everything he would want to know about mountain climbing, which is a fair and objective measure of the program's success.

The bottom line: For simulation fans.

Final Assault

Tower Toppler

by U.S. Gold
Epyx
600 Galveston Drive
Redwood City, CA 94063
(415) 366-0606
$49.95, color only

Reviewed by Steve Panak

I've been waiting for this one. Not this particular game, mind you, but rather for one very much like it. And I'm not so sure that the correct word shouldn't be "dreading." This is because I lack the free time to be hooked on something like *Tower Toppler*, the latest arcade-action addiction from Epyx.

The premise is simple enough. In the middle of a toxic ocean rise two sets of eight towers. Your goal is to climb each progressively more difficult donjon, avoiding a whole slew of passive and active hazards that confront you. However, as these dangers merely knock you down, rather than kill you outright, and because you often catch yourself before you take a dip in the deadly waters, most players will find the time limit to be their most persistent enemy.

But what keeps you masochistically coming back for more are the towers themselves. Each one is a labyrinth with breakaway trap floors, barricades and only one way to the top. And even after you find the way, you'll need split-second timing to do it consistently enough to finish all seven towers consecutively.

Truly successful arcade games require an identifiable protagonist, and in this department, *Tower Toppler* excels. True, while the small, green, Q-Bert-like creature with the huge, expressive eyes may not go down in high-resolution history as the next *Pac-Man* or *Donkey Kong*, he is just weird enough to grab your attention. And if he doesn't enthrall you, the superb graphics will. The towers rotate smoothly as you circle up them, and every item in the game is finely shaded and detailed—truly arcade quality. Control is, likewise, sure and precise, and the action is supplemented with realistic sound effects and mindless music that will hypnotize you through hours of plays. As if this isn't enough, the bonus rounds will provide just the impetus you need to complete just one more tower before powering down.

All in all, *Tower Toppler* is a primer on just what makes a great arcade program. As I mentioned earlier, you've got the cute creature, you've got danger, you've got challenge. Just keep in mind that once you boot this one up, it's unlikely that you do anything else until it is conquered. Consider yourself warned.

The bottom line: Buy it.

Tower Toppler

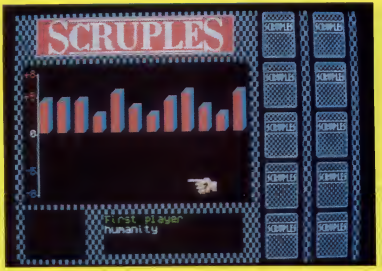
Scruples

by Milton Bradley
Electronic Arts
1820 Gateway Drive
San Mateo, CA 94404
(415) 571-7171
$39.95, color only

Reviewed by Steve Panak

It seems so long ago that the *Trivial Pursuit* fad grabbed the country. As I predicted at the time (and I must add that such clairvoyance required neither supernatural powers nor a hotline to the Almighty), the addiction to those little cards soon faded.

But not before a lot of other games based on the premise that a room full of people required a stack of such cards simply to engage in conversation or otherwise enjoy themselves popped up on department store shelves. *Scruples* was one such game.



Scruples

The object of the game is to pose a moral dilemma to a person and force him to answer it to the satisfaction of the rest of the group. The winner is the player who is able to predict who will answer each question in a given way. Such a question might be: Would you return a wallet that you found on the street, if it contained $1,000 and no one knew you had it? And while I find little pleasure in such a contest, a number of people do, and did, and more power to them. But the computer version, or should I say perversion, is another thing entirely.

What I find truly ghastly about this game is the fact that you play against computer-generated and -controlled people, infinitely inferior to real people. I'll disregard the option that allows you to play with real people, as I feel, as usual, that there is no need to insert a computer into a game such as this. It simply gets in the way. So the person most likely to play the game can best be described as follows.

Presumably the most likely player of computerized *Scruples* enjoys the board version of the game but can find no one to play *Scruples* with. Possibly he has no friends at all. So this lonely person selects a number of computer-generated "friends," who have been given ratings in 12 traits such as honesty, truthfulness and greed. This person then tries to predict how his computer friends will solve their moral dilemmas. What is even more ludicrous, these imaginary people can even argue with each other when they disagree. And you can argue with them, choosing from four canned responses.

But I'm not going to argue with *you*. If you feel that you have to play *Scruples* against a computer, run right out and get this game. However, I feel that most players will immediately realize that the main ingredient of this game has been removed: the infinite complexity of human personalities and relationships.

And with this removed, there is little left.

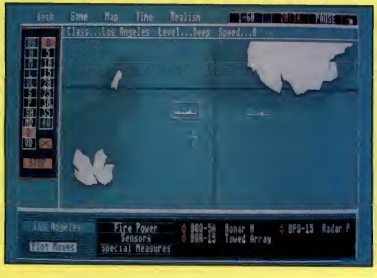
The bottom line: Skip it.

Under the Ice

Lyric Software
6 Beach Plum Drive
Northport, NY 11768
(516) 754-5570
Color only

Reviewed by Robert Goff

Under the Ice



One of the problems with modern military simulations is that even if you could be accurate without compromising military secrets, it would be excruciatingly boring. Real soldiers and pilots and sailors spend years preparing for a few minutes of battle. Those few minutes won't really be much fun (or survivable) without the years of preparation. So we have conflicting goals: to realistically simulate something that isn't really fun in a fun and exciting way. I don't think that *Under the Ice* has found the balance.

Under the Ice is a single-player game that puts you in the role of a task force commander (either NATO or Soviet) with two or more submarines under your direct command. You pick the scenario and the side you fight on, the computer picks your ships and your weapons so that each game is a little different. Your mission is to use your ships to find and sink the enemy before he does the same to you.

There are a variety of ship classes and weapons. Permit, Sturgeon, Los Angeles and Trafalgar class ships on the NATO side and Victor 1, Victor 3, Yankee and Delta class ships on the Soviet side. Different ships carry different combinations of torpedoes and missiles. You track the progress of your units on a map of their patrol area with sensor information provided either as lines pointing from the detecting platform toward the contact or, if the information is good enough, as icons representing the location of the enemy ship. To give orders to one of your ships, you click on it and use the control buttons to control weapons, sensors or movement. To get information about a target, you click the mouse on the target on the screen. A line at the top of the map tells you the class of ship, its depth and its speed. After you select a target, you can order one of your ships to shoot it.

The game disk itself is unprotected. For copy protection, it uses a password system where the game asks you to type in a word from a random place in the manual. This method is certainly a lot less painful than disk encryption, since it leaves you able to make your own backups.

The play of the game is fair. Selections under the GEM menu bar allow pausing the game and changing the time progression.

There is no game save, and there doesn't seem to be any way to restart the game without actually quitting the program and running it again. There's also a game realism switch that determines whether you see torpedoes fired at you plotted on the map or only from the direction they are coming. The use of the mouse to select units to command and targets to shoot is a good idea, but can be a pain. If you try to select one of the ship icons on the map but miss by a little, you execute a map zoom.

If you're used to animation that moves smoothly, or even semi-smoothly, the complete redraw of the screen every two seconds will annoy you. Having to wait for the program to accept commands may also drive you nuts. But to be fair, this simulation doesn't really have to have the instant response of an arcade game.

While controlling the movement of the ships by entering route legs is convenient, there doesn't seem to be a way to change the current depth or speed without deleting all the route legs back to your present position and entering new ones. Very clumsy. The time the ship reaches the end of each route leg is listed on the screen to allow coordinating task force movement, but since the time isn't shown until the leg is defined, it's not very useful.

There are problems in the information line that give depth and speed at the top of the map. If you change depth or speed, it doesn't update until you select another ship and then go back to the first.

The simulation is less than fair. The capabilities of the different classes of submarines are not fairly represented by the speeds and depths at which they are allowed to operate. I can only hope that their noise levels and sonar capabilities are modeled in the program to provide a realistic advantage of one ship over another. The arena of action, a small section of the North Sea, is too limited. Finally, the manual attempts to give some background and hints on tactics, but doesn't go far enough. It concentrates on how sound propagates in the water without telling you how to use that information.

This game has many of the strategic aspects of a war game and less of the shoot-'em-up action of the video game. If you like a game that is more tactics than reflexes, you may enjoy this one. If you're looking for an accurate submarine simulation, however, save your money. ■

The bottom line: Get a demonstration first.

Frank Eva is an auditor by profession, but has been involved in the computer industry ever since his purchase of an Atari 400 many years ago. He has dabbled in programming and has had several text adventures published.

Steve Panak has written more game reviews for ST-LOG *and ANALOG than anyone on the face of the earth. He lives in Ohio where he plays games on his ST and practices law.*

Robert Goff is a part-time freelance author specializing in the Atari ST. To pay the bills, he is a naval officer with ten years of experience in submarines.

The ST-LOG #33 diskette contains 28 magazine files. They are listed below.

JULY 1989 · DISK LISTINGS

```
FILENAME.EXT   FILE TYPE   COMMENTS
------------   ---------   --------

CMANSHIP.ARC containa:
  C33     .PRG   RUN FILE    G-MANSHIP COMPILED
  C33     .C     G           C-MANSHIP LISTING

ANIMINPT.ARC containa:
  ANIMINPT.BAS   GFA BASIC   ANIMATED INPUT
  ANIMINPT.LBT   GFA BASIC   ANIMATED INPUT, ASCII

DRAWITCO.ARC containa:
  DRAW_IT .PRG   RUN FILE    DRAW IT! COLOR VER.
  DRAW_IT .LST   GFA BASIC   SOURCE CODE, ASCII
  SHOW_IT .PRG   RUN FILE    SHOW IT! COLOR VER.
  SHOW_IT .PRG   GFA BASIC   SOURCE CODE, ASCII

DRAWITMO.ARC containa:
  MDRAW_IT.PRG   RUN FILE    DRAW IT! MONO. VER.
  MSHOW_IT.PRG   RUN FILE    SHOW IT! MONO. VER.

GAMECUPB.ARC contains:
  CUPBOARD.PRG   RUN FILE    ST GAME CUPBOARD
  CUPBOARD.PI1   DEGAS       PICTURE FILE
  EIGHT   .PI1   DEGAS       PICTURE FILE
  NOUGHT  .PII   DEGAS       PICTURE FILE
  REPEAT  .PII   DEGAS       PICTURE FILE
  ANSWERS .EIT   DATA        GAME DATA FILE

GUESSSKH.ARC contains:
  GUESS   .PRG   RUN FILE    GUESS-A-SKETCH
  HOWTOPLA.DOC   TEXT        GAME INSTRUCTIONS
  (OTHER FILES IN THIS ARC ARE DATA FILES
     REQUIRED BY THE GAME.)

MICROCHK.ARC containa:
  MICROCHK.PRG   RUN FILE    MICROCHECK ST
  MICROCHK.RSC   RESOURCE    RESOURCE FILE
  MICROCHK.C     G           SOURCE CODE
  MICROCHK.H     C           HEADER FILE
  MICROCHK.DFN   DATA        RCS DATA FILE

TIMER.ARC contains:
  TIMERS  .PRG   RUN FILE    MILLISECOND TIMER
  TIMERS  .C     C           SOURCE CODE
  TIMERS  .S     ASSEMBLY    SOURCE CODE
  INTR    .S     ASSEMBLY    SOURCE GODE
  BI2     .BAT   TEXT        BATCH FILE
  BINT    .BAT   TEXT        BATCH FILE

  README  .DOC   TEXT        Diak instructiona
  UNARCHIV.DOC   TEXT        Unsrchiving inatructions

Disk instructions:
   Only those files with .PRG, .TOS, or .TTP
   extensions may be run from tbe GEM desktop.
   Other programs may require additional software
   as shown below.  The filea on thia disk have
   been archived (compressed) into .ARC files.  To
   reatore the programs to their runnable form,
   follow the instructions found in the
   UNARCHIV.DOC file.

   NOTE: due to space limitations, the source code
   for Game Cupboard and for the high res version
   of Drsw It! are not on this disk.  If you wish
   to obtain these files, send your originsl July
   '89 disk (after making s copy; the original will
   be erased), along with a postage-paid mailer to:

        ST-LOC JULY SOURCE CODE
        P.O. BOX I413-M.O.
        MANCHESTER, CT  06040-1413

WARNING: Be sure to read the appropriate magazine
         srticle before attempting to run magazine
         files.  Failure to do so may yield
         confusing results.

                .EXT  DESCRIPTION
                ----  -----------
                .BAS  Requirea GFA BASIC
                .LST  Requires GFA BASIC
                .G    Requires C compiler
                .S    Requires 68000 assembler
                .PAS  Requires Paacal compiler
```

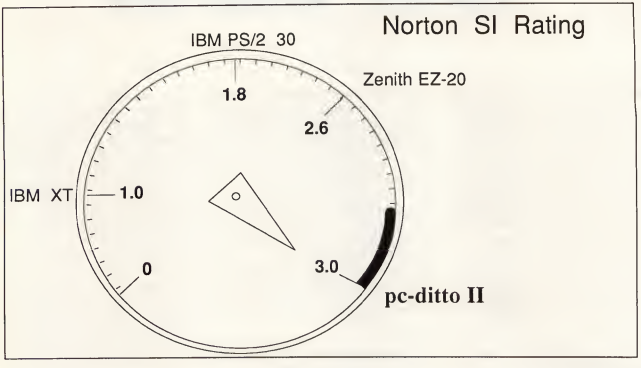
ST LOG

INDEX TO ADVERTISERS

This index is an additional service. While every effort is made to provide a complete and accurate listing, the publisher cannot be responsible for inadvertent errors.

pc-ditto II

By Avant-Garde Systems



IBM **XT** **COMPATIBLE** !

IBM **AT** **PERFORMANCE** !

ATARI **ST** **PRICE** !

$299.95

Now! Run the most popular IBM software on your Atari ST...

FAST !

See pc-ditto II at your local Atari dealer or write for free information.

Yes. Please send me more information !

Avant-Garde Systems, 381 Pablo Point Drive Jacksonville, Florida 32225

Name_____

Address_____

City_____St_____Zip_____

CIRCLE #112 ON READER SERVICE CARD.

Show Me Your Wares

by Karl E. Wiegers

It used to be a store that sold hardware was someplace you went to buy nails or to pick up a pane of glass for the house. But times change, and now when you talk about a hardware store, you just might be referring to the nearby ComputerLand. The term hardware has grown to encompass computers, peripherals like monitors and printers, the boards you plug into slots in your computer, and even the electronic components on the boards.

Words are funny. When you have an expression that contains both an adjective and a noun, you can envision a corresponding expression containing an adjective having the opposite meaning. For example, there was no such thing as an "acoustic guitar" until someone invented the "electric guitar." In this case, I'm thinking of the word software. Software has come to refer to the stuff you run on a computer, whether it comes from a tape, a disk drive, or (if you date back to the computer Stone Age) even from switches on the computer's front panel. Now you can actually go to a "software store," which is sort of a white-collar equivalent of the old-time hardware store.

During my reading of computer literature, I've encountered all sorts of other computerware words. It seems to be mighty fashionable to append the suffix "ware" to practically any other word to create something that sounds both clever and computerish. I thought you might be interested in hearing some of these.

The words "hardware" and "software" generate very different images in our minds. Hardware is tangible, concrete and substantial. It has some heft to it. Software seems more pliable, ephemeral. Not surprising, when you figure that software consists mostly of some little magnetized specks on a piece of plastic. If you look at a floppy disk, you can't tell if there's any software on it or not.

But what comes in between? *Firmware*, of course. Firmware is basically software in which you have a lot of confidence. You create firmware by writing some software and then storing it for eternity in the silicon of a ROM chip. No more transient magnetic blips for this program! The downside is that you can't change the contents of the ROM, which is why you need a lot of confidence in your software before firming it up. If you have a bit less confidence, you might store it in an EPROM, an erasable, programmable, read-only, memory chip. It's more convenient than a floppy disk, but reusable if your lack of confidence proves premature.

Then there are those companies who try to get you all excited about computer products that they promise but never deliver. Such imaginary stuff is called *vaporware*. Atari Corporation has practically made an art form of vaporware, with substantial offerings in both the hard and soft vaporware categories. Vaporware is not limited to the computer world, but there sure seems to be more of it there (or not there, depending on how you look at it) than for other industries.

Some kind souls don't try to sell you anything at all—they give it away. The contributions of these generous folks are referred to as *freeware*. Public domain programs are in the freeware category. While there's an awful lot of good freeware available, remember that sometimes you get what you pay for.

The next best thing to freeware is *shareware*. This is the best "ware" word because both syllables rhyme. The authors of shareware have a sensible attitude. Basically, they're saying, "Here. Check this out. If you like it, pay me what you feel it's worth to you." Donations in the $10 to $20 range are usually suggested. This is a registration fee, which may entitle you to future upgrades, printed manuals, etc.

Shareware authors are trying to get some financial reward for their efforts without going through the expensive and iffy route of commercial publication. You're encouraged to give copies of shareware to your friends, who can evaluate it and send in their registration fees if they use the program. Don't confuse shareware and freeware. Shareware shouldn't be considered public domain, so if you use it, send the guy a few bucks.

The sad fact is that not every program you acquire turns out to be useful. Such products become transformed into *shelfware*. They can be found sitting quietly on your shelves, sometimes still in the original shrink-wrap. I've heard you can determine the age of shelfware programs by counting the rings in the dust they've collected (okay, I made this part up).

Sometimes programs are made available to a select group of users before being officially released. The idea is to test them thoroughly, so that the errors show up in friendly hands. This is much less traumatic than having genuine customers who paid real money discover the bugs. Since this second-stage testing is called "beta testing," the products naturally are referred to as *betaware*. Product reviewers often get a hold of betaware, so they have to be careful to inform the reader that they weren't testing the official, presumably bug-free (ha!) release of the program. Programs that don't survive the betaware experience may metamorphose into vaporware.

Sometimes you'd like to try out a program before plunking down your cash, especially for products in the multi-hundred-dollar price range. Some vendors make this possible by selling at a low price or giving away restricted versions of the package. For example, a database program may let you set up only a tiny data-

base with 20 records or so: enough to see how it operates, but not enough to do useful work. You get the complete program when you've paid the full price. Shareware can work the same way, with the full features being enabled somehow after your registration fee is received. I've heard such limited versions of programs referred to as *crippleware*.

As computer technology advances, the nature of the supporting software evolves. Many organizations are linking their personal computers through local area networks. Of course, the specialized software that runs on networks has to be called *netware*. (NetWare is also a registered trademark of Novell, Inc., which sells products for local area networks.)

One benefit of networking your personal computers is enhanced communication among human beings. Software that facilitates the interactions of a number of people is called *groupware*. Computer conferencing might be one kind of groupware. A word processor that somehow lets people work together on the same document simultaneously would also qualify as groupware.

People are finding all sorts of ways to automate their daily activities. Unfortunately, most available software addresses just one isolated function, rather than being an integrated solution to a complex job process. If you want to automate a sequence of activities, you probably have to pass data from one program to another. Software tools to accomplish such linking are called *bridgeware*. Some vendors will show you horrendous diagrams depicting the clumsy use of bridgeware in a feeble attempt to automate your entire work process. Then they present their elegant integrated system that makes your tangled mess obsolete—and for the low, low price of just $200,000. (I actually saw just such a vendor presentation, with that very price.)

Another hot topic in computing these days is hypertext, and programs based on hypertext are known as *hyperware*. Perhaps the most popular example of hyperware is Apple's HyperCard for the Macintosh. Hypertext is an unusual kind of database, in which you store information on a sort of electronic index card. A database consisting of a bunch of cards is called a "stack," so another term for hyperware is *stackware*.

This about covers the different sorts of computer wares I have encountered in my readings. But the single most important thing to remember in your computer travails is that old Latin expression *caveat emptor*: let the buyer be-ware. ▪

Karl Wiegers *is a software engineer in the Eastman Kodak Photography Research Laboratories. Although he is obviously well read on the subject of "wares," he overlooked the most important type: the wonderful programs that can be found on this magazine's disk version, which are referred to, of course, as "logware."*

ST-LOG *invites all authors to submit essays for possible use in the* Footnotes *department. Submissions should be between 1,000 and 1,500 words and may be on any aspect of Atari computing. Any style or type of essay is acceptable—opinion, humor, personal experience—but creativity is a plus. Send your submission to:* Footnotes, *c/o* ST-LOG, *P.O. Box 1413-MO, Manchester, CT 06040-1413.*

(from page 59)

artist's sketch pad as the metaphor and each page (screen) can have its own palette. They also provide pens, pencils, markers, chalk, stump, tracing tools, brushes and similar artist's tools. It's an unusual way to work but rather more intuitive if your perception hasn't been spoiled by other art programs that more or less ignore the graphic arts standards in their interface.

Rough has several uncommon commands, including the viewing of black lines only, breakdown of a painting ("page") into black, yellow, blue and red (for producing color separations), animation of pages, 3-D arrows, flexible ruler (for curves), merge sketches and mask parts of sketches. It also has GDOS support. More remarkable is *Rough*'s ability to handle CAD 3-D V2 objects—retrieve, manipulate and copy the image to a sketch page. This almost allows *Rough* to be a combination draw/paint program!

Rough's file-handling capability includes *Neochrome*, *DEGAS* regular and compressed, *Art Director* and *GFA Artist* formats.

Rough is the best graphic-arts paint program I've encountered, and well worth getting when it comes, translated, across the ocean. If bundled with its sister program, *Lazy Paint*, it will be a dynamite package. Unfortunately, I've only seen them as beta English-language versions, but what they've done so far is impressive and exciting. There is a lot of good software in Europe.

In the meantime, I'm going to continue to doodle with my artware—maybe even find a few more to compare—and play that highly addictive game, *Tetris*, from Spectrum Holobyte. But more on that next time.

Ian Chadwick *is a Toronto-based technical writer who lives in an increasingly small house with his wife, Susan, six cats, one dog, two rats and several field mice (who moved in recently, despite the cats)—and that's not to mention the neighborhood's stray cats that take up residence as the mood moves them.*

SMASH! HITS!

Looking for some real action in home computer games? Excitement that'll have your heart pounding and palms sweating? Razzle dazzle graphics that'll drive you wild?

Look no further than these incredible arcade smash hits — now *faithfully* converted by Tengen.

Speaking of smash, Blasteroids® gives a lot of space rocks a chance to do exactly that to your starship. Twist, dodge and blast away at the never-ending onslaught of asteroids and enemy ships. But one wrong move and you'll bite space dust.

When it comes to hits, nothing lets you give or take more than Vindicators.™ It puts you in charge of a hyper futuristic tank that'll blow your mind — fighting against enemy tanks and turrets that'll blow you *away*.

Get Tengen's smash hits at your favorite retailer today. And experience the best of the arcades at home!

They're available now for the C-64/128, Amiga and Atari ST.

TENGEN

WE BRING THE BEST ARCADE HITS HOME.

1901 McCarthy Boulevard., Suite 210, Milpitas, CA 95035 (408) 435-2650

CIRCLE #114 ON READER SERVICE CARD.

A CATACLYSMIC STRUGGLE BETWEEN GOOD AND EVIL

For the first Time, Tolkien's panoramic vision of the cataclysmic struggle between good and evil has been skillfully crafted into a single computer game of epic proportions.

FIVE MAN-YEARS of detailed research and programming have been invested to ensure that this is the "definitive Tolkien computer game." Selected elements of traditional Fantasy Role Playing, War strategy, and animated adventure games have been skillfully blended for a unique computer gaming experience.

Follow in the footsteps of Frodo, Aaragorn and Gandalf as they battle to get the ring to the Cracks of Doom. Ranged against you are all the evil forces of the Dark Lork Sauron and the corrupt wizard Saruman. The odds are overwhelming, but you cannot afford to fail. The destiny of Middle Earth lies in the balance.

Roam at will across the 36-screen scrolling map of Middle Earth, wherein are contained thousands of digitized locations in which you can control a cast of over 80 animated characters and armies. Test your strategic abilities as you command entire legions to move against the forces of evil.

With the almost infinite variety and multiple layers of play, War in Middle Earth is a game you can finish in days or weeks, or savor for months. It will delight and enthrall, and ultimately become a firm favorite in any software library.

May your heroic efforts be met with success, and the forces of darkness be vanquished!

J.R.R. tolkien's

WAR IN MIDDLE EARTH

NOW SHIPPING FOR:
IBM PC, PS/2, Tandy 1000 (5-¼" CGA/EGA)
IBM PC, PS/2, Tandy 1000 (3½" CGA/EGA)
IBM PS/2 (3-½" MCGA) • Amiga • Apple IIGS • Atari ST

SUGGESTED RETAIL: $49.99

SCREEN SHOTS TAKEN FROM AMIGA, IBM, APPLE IIGS AND ATARI ST VERSIONS.

Melbourne House
711 West 17th St.,
Unit G9, Costa Mesa,
CA 92627 · (714) 631-1001

MELBOURNE HOUSE

CIRCLE #115 ON READER SERVICE CARD.